Extracción de preguntas a partir de imágenes para personas con problemas de memoria mediante técnicas de Deep Learning - Question extraction from images for people with memory problems using Deep Learning techniques

Trabajo de Fin de Grado
Curso 2020–2021

Autores
Alejandro Aizel Boto
Roberto Portillo Torres
Daniel Sanz Mayo

Director
Alberto Díaz Esteban

Facultad de Informática
Universidad Complutense de Madrid

Extracción de preguntas a partir de imágenes para personas con problemas de memoria mediante técnicas de Deep Learning - Question extraction from images for people with memory problems using Deep Learning techniques

Trabajo de Fin de Grado

Autores
Alejandro Aizel Boto
Roberto Portillo Torres
Daniel Sanz Mayo

Director
Alberto Díaz Esteban

Convocatoria: *Junio 2021*

Facultad de Informática
Universidad Complutense de Madrid

2 de julio de 2021

Autorización de difusión

Los abajo firmantes, matriculados en el grado de Ingeniería del Software e Ingeniería Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores el presente Trabajo Fin de Grado: .Extracción de descripciones de recuerdos a partir de material multimedia mediante técnicas de Deep Learning", realizado durante el curso académico 2020/2021 bajo la dirección de Alberto Díaz Esteban en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Alejandro Aizel Boto
Roberto Portillo Torres
Daniel Sanz Mayo

2 de julio de 2021

Dedicatoria

Alejandro Aizel Boto

Me gustaría dedicar este proyecto a mi familia. Primero a mis padres que siempre me han apoyado y me han dado lo que necesitaba. A mi tía, que desde pequeño me ha enseñado todo. Soy como soy gracias a ella. Y finalmente a mi hermana, que se que siempre está cuando la necesito y sé que así seguirá siendo. Os quiero.

Roberto Portillo Torres

Me gustaría dedicar este proyecto a mis amigos Clara, Ignacio y Fernando, gracias a ellos me levanto cuando caigo, a mi hermana, la persona a la que más admiro en el mundo. También a mi madre y a mi padre por estar a mi lado cuando todo se derrumba. Gracias por tanto, nunca os olvido.

Daniel Sanz Mayo

Me gustaría dedicar este proyecto a mis padres, que me han apoyado siempre en todo lo que he hecho y a mi abuela que siempre se preocupa por mi y me anima a seguir cuando peor estoy. También a mis amigos por ayudarme a desconectar de todo cuando les veo. Sin ellos esto no habría sido posible.

Agradecimientos

A nuestro tutor *Alberto Diaz Esteban* por ayudarnos con todas las dudas que hemos ido teniendo a lo largo del curso, responder nuestros correos rápidamente y por animarnos cada día. Gracias por todo.

Resumen

El Alzheimer es una enfermedad que afecta a la memoria, el pensamiento y el comportamiento[1]. Se estima que entre el 60 % y el 70 % de las personas con demencia a nivel mundial sufren Alzheimer[2].

Este proyecto nace con el objetivo de ayudar a estas personas a recordar aspectos de su vida utilizando técnicas de *Deep Learning* como redes convolucionales y recurrentes. Para lograrlo desarrollamos un sistema capaz de extraer preguntas de fotografías que puedan representar recuerdos para las personas con problemas de memoria utilizando un *bot* que simula una sesión de terapia de reminiscencia.

El usuario tendrá que enviarle las fotografías al *bot* y este se encargará de enviarle, una a una, las preguntas generadas por la red neuronal. En este momento, el usuario deberá recordar todo lo posible sobre la imagen para poder responder a las preguntas y conseguir ejercitar su memoria.

La reminiscencia es una técnica que favorece la evocación de recuerdos y sucesos del pasado para conectarlos con el presente y fortalecer así la propia identidad de la persona frente a esta enfermedad[3].

Todos los recursos del proyecto, ejecutables y código se pueden encontrar en:

https://github.com/NILGroup/TFG2021RecuerdosMultimedia

[1] ¿Qué es el Alzheimer?: https://www.alz.org/alzheimers-dementia/what-is-alzheimers

[2] Información sobre el Alzheimer: https://www.mayoclinic.org/es-es/diseases-conditions/alzheimers-disease/symptoms-causes/syc-20350447

[3] Terapias de reminiscencia para personas con demencia: https://orpea.es/terapia-de-reminiscencia-para-personas-con-demencia/

Palabras clave

Deep Learning, Reminiscencia, Convolutional Neural Network, Recurrent, Neural Network, Visual Question Generation, Bot de Telegram, Encoder, Decoder.

Abstract

Alzheimer's is a disease that affects memory, thinking, and behavior. It is estimated that between 60% and 70% of people with dementia worldwide suffer from Alzheimer's.

This project was born with the aim of helping these people to remember aspects of their lives using *Deep Learning* techniques such as convolutional and recurrent networks. To achieve this we developed a system capable of extracting questions from photographs that can represent memories for people with memory problems using a *bot* that simulates a reminiscence therapy session.

The user has to send the photographs to the *bot* and the *bot* will be in charge of sending him, one by one, the questions generated by the neural network. At this time, the user must remember as much as possible about the image in order to answer the questions and be able to exercise his memory.

Reminiscence is a technique that favors the evocation of memories and events of the past to connect them with the present and thus strengthen the person's own identity in the face of this disease.

All project's resources, executable files and code can be accessed from:

https://github.com/NILGroup/TFG2021RecuerdosMultimedia

Keywords

Deep Learning, Reminiscence, Convolutional Neural Network, Recurrent, Neural-Network, Visual Question Generation, Telegram Bot, Encoder, Decoder.

Índice

Índice de figuras

Capítulo 1

Introducción

"Cambiaría toda mi tecnología por una tarde con Sócrates"
— Steve Jobs

1.1. Motivación y Objetivos

Debido al aumento de la esperanza de vida durante las últimas décadas, el número de casos de enfermedades neurodegenerativas como el Alzheimer ha aumentado considerablemente. El Alzheimer es una enfermedad que se caracteriza por la pérdida de memoria inmediata y de otras capacidades mentales, como las cognitivas superiores, al tiempo que se van muriendo las neuronas y se van atrofiando determinadas partes del cerebro[1].

Actualmente no se dispone de ningún tratamiento que cure el Alzheimer, por lo que los tratamientos actuales se utilizan para paliar los síntomas y aumentar la actividad de las neuronas con el objetivo de minimizar los efectos degenerativos[2].

Además de no existir tratamientos, tampoco es fácil para muchos pacientes acceder a estos procedimientos. Estas terapias suelen ser costosas ya que requieren de la presencia de terapeutas, y muchas veces el traslado a hospitales o residencias especializadas para poder realizarlas.

Creemos que, al igual que las nuevas tecnologías han contribuido a la hora de mejorar el nivel de vida de las personas en ámbitos como la ciencia y la sanidad, la

[1] Información sobre el Alzheimer: https://es.wikipedia.org/wiki/Enfermedad_de_Alzheimer
[2] Información sobre los tratamientos de las enfermedades neurodegenerativas: https://neurorhb.com/enfermedades-neurodegenerativas/

inteligencia artificial puede ayudar a las personas con problemas neurodegenerativos a acceder a sistemas innovadores que ofrezcan nuevas soluciones a las necesidades de los pacientes, así como ayudar a los familiares y cuidadores a comprender la enfermedad y monitorizar su progreso.

Se calcula que en España hay más de 1.150.000 familias afectadas por Alzheimer, Párkinson, Esclerosis Múltiple, enfermedades neuromusculares y Esclerosis Lateral Amiotrófica. En concreto, 1 de cada 2.000 personas presentan Enfermedades Neuromusculares. El 10 % de las personas mayores de 65 años y el 50 % de las personas mayores de 85 años tiene Alzheimer en nuestro país. Además, esto trae un coste a las personas afectadas y a sus familiares de más de 23.000 € al año[3].

Uno de los tratamientos más utilizados para prevenir el Alzheimer es lo que se conoce como terapia de Reminiscencia. Consiste en mantener una conversación con el paciente y hacer que este piense sobre su propia experiencia vital con el fin de que consiga recordar y reflexionar sobre su pasado. La terapia se puede realizar de manera formal, con un profesional o terapeuta, o más informal, con familiares o amigos[4].

Debido a las situaciones descritas anteriormente, nuestra motivación es utilizar técnicas de *Deep Learning* (DL) para desarrollar herramientas que puedan ayudar a las personas que se encuentren en las primeras fases de esta enfermedad. El principal objetivo de este proyecto es realizar un sistema capaz de generar preguntas extrayendo características de imágenes de la vida de la persona en cuestión y almacenarlas para después poder utilizarlas, ayudando así en el tratamiento de su enfermedad.

Para alcanzar nuestro objetivo dividiremos nuestro trabajo en diferentes fases:

1. Encontrar conjuntos de datos libres y *open-source* con imágenes y preguntas para poder entrenar nuestros modelos.
2. Explorar las técnicas utilizadas para la generación de preguntas visuales, *Visual Question Generation* (VQG).
3. Realizar un *bot* en Telegram que permita la interacción con el paciente lo más intuitiva posible y natural al tratarse de una aplicación de mensajería.
4. Integrar ambos modelos para simular la terapia de reminiscencia.

[3]Información y datos sobre las personas afectadas por las enfermedades neurodegenerativas: http://neuroalianza.org/las-enfermedades-neurodegenerativas/que-son/

[4]Información sobre la terapia de reminiscencia: https://www.estimulacioncognitiva.info/2018/04/11/qué-es-la-terapia-de-reminiscencia-y-cómo-se-puede-realizar/

Chapter 1

Introduction

"I would trade all of my technology for an afternoon with Socrates"
— Steve Jobs

1.1. Motivation and Goals

Due to the increase in life expectancy during the last decades, the number of cases of neurodegenerative diseases such as Alzheimer's has increased considerably. Alzheimer's is a disease characterized by the loss of immediate memory and other mental abilities, such as higher cognitive abilities, while neurons die and certain parts of the brain atrophy [1].

Currently there is no treatment available to cure Alzheimer's, so current treatments are used to alleviate symptoms and increase the activity of neurons in order to minimize the degenerative effects[2].

In addition to the absence of treatments that cure these types of diseases, it is not easy for many patients to access these procedures. These therapies are usually expensive because they require the presence of specialized therapists, and also the transfer to hospitals or specialized residences to be able to perform them.

We believe that just as new technologies have contributed to improving people's living standards in areas such as science and healthcare, artificial intelligence can

[1] Alzheimer's information: https://es.wikipedia.org/wiki/Enfermedad_de_Alzheimer

[2] Information about treatments for neurodegenerative diseases: https://neurorhb.com/enfermedades-neurodegenerativas/

help people with neurodegenerative problems to access innovative systems that offer new solutions to patients' needs, as well as help family members and caregivers to understand the disease and monitor its progress.

It is estimated that in Spain there are more than 1,150,000 families affected by Alzheimer's, Parkinson's, Multiple Sclerosis, Neuromuscular Diseases and Amyotrophic Lateral Sclerosis. Specifically, 1 in 2,000 people have Neuromuscular Diseases. 10 % of people over 65 and 50 % of people over 85 have Alzheimer's in our country. In addition, this entails a cost to the affected people and their families of more than € 23,000 per year [3].

One of the most used treatments to prevent Alzheimer's is known as Reminiscence therapy. It consists of having a conversation with the patient and making him think about his own life experience in order for him to be able to remember and reflect on his past. Therapy can be done formally, with a professional or a therapist, or more informally, with family or friends [4].

Due to the situations described above, our motivation is to use DL techniques to develop tools, more specifically an intuitive dialogue system, that can help people who are in the early stages of this disease. The main goal of this project is to create a system capable of generating questions by extracting characteristics from images of the life of the patient and storing them for later use, helping in the treatment of their disease.

To achieve our goal we will divide our work into different phases:

1. Find free and *open-source* databases with images and questions to train our models
2. Explore the techniques used to generate visual questions (VQG).
3. Create a *bot* in Telegram that allows interaction with the patient as intuitive and natural as possible as it is a messaging *app*.
4. Integrate both models to simulate reminiscence therapy.

[3] Information and data about people affected by neurodegenerative diseases: http://neuroalianza.org/las-enfermedades-neurodegenerativas/que-son/

[4] Information about Reminiscence therapy: https://www.estimulacioncognitiva.info/2018/04/11/qué-es-la-terapia-de-reminiscencia-y-cómo-se-puede-realizar/

Capítulo 2

Deep learning y trabajo previo

Realizar un proyecto como este es complicado y se necesitan de herramientas potentes como el DL para poder resolverlo. Para la realización del proyecto utilizamos tecnología de DL y en concreto lo que se conoce como VQG, es decir, generar preguntas en base a fotografías. La idea está basada en el Paper **[Automatic Reminiscence Therapy for Dementia]**[1] de Mariona Carós en el que utilizan técnicas de DL para generar una sesión de terapia interactiva con un paciente. Al igual que Mariona, utilizamos Telegram para poder implementar nuestra aplicación.

Durante el inicio del curso y las primeras semanas de trabajo en nuestro proyecto, todos los miembros del grupo trabajamos en aumentar y complementar nuestros conocimientos sobre las diferentes herramientas, conceptos y tecnologías que íbamos a utilizar durante este. Vamos a comenzar a explicar los conceptos de DL que hemos necesitado para poder afrontar este proyecto.

2.1. Deep Learning

El término Inteligencia Artificial (IA) fue adoptado en 1956, pero se ha popularizado durante estos últimos años gracias al incremento en los volúmenes de datos, algoritmos avanzados, y mejoras en el poder de cómputo y el almacenaje. Podríamos decir que la IA es la inteligencia llevada a cabo por máquinas. Una de las ramas más populares actualmente es el *Machine Learning* (ML) y más específicamente el DL.

[1] Automatic Reminiscence Therapy for Dementia: https://arxiv.org/pdf/1910.11949.pdf

2.1.1. ¿Qué es el Deep Learning?

El DL es un campo del ML en el que se utilizan algoritmos mucho más grandes y potentes con el fin de obtener resultados cada vez más cercanos a los que obtendría un humano. Se trata de un conjunto de redes neuronales, con tres o más capas, que intentan simular el comportamiento del cerebro humano (aunque lejos de igualar su capacidad) con el fin de aprender grandes cantidades de datos[2].

El DL ha traido numerosas aplicaciones y servicios inteligentes que mejoran la automatización, realizando tareas analíticas y físicas sin la intervención humana. Esta tecnología se encuentra detrás de numerosos servicios y productos cotidianos como el control por voz, detección de rostros y de tecnologías emergentes como la conducción autónoma[3].

2.1.2. Cursos Deep Learning

Lo primero que hicimos fue el curso especializado de DL de Andrew Ng (6) en Coursera [4]. Aunque ya teníamos conocimientos sobre ML, había muchos conceptos que no conocíamos por lo que estos cursos fueron una gran fuente de conocimiento.

Este programa especializado está dividido en cinco cursos bien definidos:

- **Redes neuronales y aprendizaje profundo**: Este primer curso nos sirvió como introducción y repaso de los conceptos más importantes sobre las Redes Neuronales (RN).
- **Mejora de las redes neuronales profundas. Ajuste, regularización y optimización de hiper-parámetros**: En este curso conocimos nuevas técnicas para mejorar nuestras RN que nos ayudaron muchísimo a la hora de mejorar nuestra implementación en el proyecto.
- **Estructuración de proyectos de Machine Learning**: Este curso nos permitió saber qué hacer en caso de que nuestra implementación no funcionase y aprendimos técnicas para poder hacer frente a estos problemas.
- **Redes Neuronales Convolucionales (CNN)**: En este curso se explica en detalle el funcionamiento de las Redes Neuronales Convolucionales (RNC), algo de gran utilidad para nosotros, ya que se utilizan a la hora de implementar tareas de reconocimiento, detección visual aplicadas a imágenes, vídeos y material multimedia.

[2]Qué es el Deep Learning: https://www.ibm.com/cloud/learn/deep-learning
[3]Aplicaciones del Deep Learning: https://www.mathworks.com/discovery/deep-learning.html
[4]Curso Andrew NG https://www.coursera.org/specializations/deep-learning

- **Modelos secuenciales**: Este curso, junto con el anterior, fue el más importante de los cinco ya que explica en detalle conceptos clave de las Redes Neuronales Recurrentes (RNR) y sus variantes (GRUs, LSTM) las cuales son utilizadas en chatbots, traductores automáticos, etc.

Tras finalizar estos cursos ya podíamos utilizar todos los conceptos que habíamos aprendido y que eran fundamentales para poder realizar nuestro proyecto. A continuación explicamos estos conceptos de forma general para poder entenderlos.

2.2. Redes Neuronales

Las RN son un modelo que intentan imitar el funcionamiento del cerebro humano. Están formadas por un conjunto de nodos, conocidos como neuronas, que están conectadas y transmiten señales entre sí. Estas señales se transmiten desde la entrada hasta generar una salida.

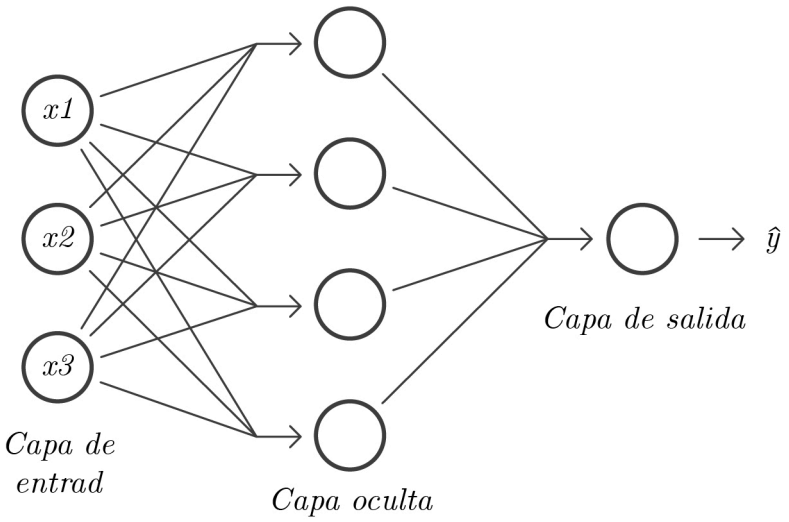

Figura 2.1: Red neuronal tradicional

Una RN puede estar formada por únicamente una neurona o por muchas neuronas. Para poder entrenar estas RN necesitamos una colección muy grande de datos. Además, a lo largo de la historia han ido apareciendo nuevas RN más sofisticadas, pensadas para resolver aquellos problemas que las RN convencionales no eran capaces de resolver. Los distintos tipos de RN que hemos utilizado en nuestro proyecto

son:

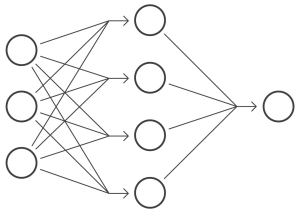

Figura 2.2: Red Neuronal convencional (RN)

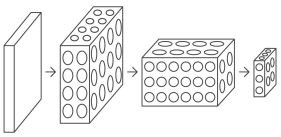

Figura 2.3: Red Neuronal Convolucional (RNC)

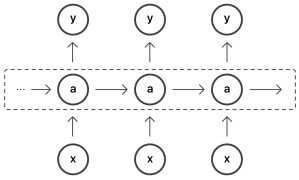

Figura 2.4: Red Neuronal Recurrente (RNR)

2.2.1. Redes Neuronales Convolucionales

La visión por computador o *Computer Vision* (CV) es una de las aplicaciones que más rápidamente está creciendo gracias al DL. Estos cambios tan rápidos que estamos viendo han permitido que aparezcan nuevas aplicaciones que no eran posibles hace unos años. Algunos ejemplos que hemos ido viendo a lo largo de los años son clasificadores de imágenes, detección de objetos en imágenes, traslado del estilo de una imagen a otra, etc. Esto ha permitido crear numerosas aplicaciones como detección de rostro, conducción autónoma, etc.

El problema que encontramos con la visión por computador es que las imágenes a veces son tan grandes que intentar resolver un problema, como los mencionados anteriormente, utilizando RN convencionales, es prácticamente imposible. Tener una imagen muy grande hace que los parámetros crezcan muchísimo. Por ejemplo si tenemos una imagen de 1.000 x 1.000 píxeles y 3 canales *Red Green and Blue* (RGB) tendríamos 1.000 * 1.000 * 3 parámetros y solamente en la primera capa. Esto computacionalmente es demasiado costoso y no podría aplicarse en el día a día. Una solución a esto es construir lo que se conoce como capas convolucionales en lugar de capas completamente conectadas.

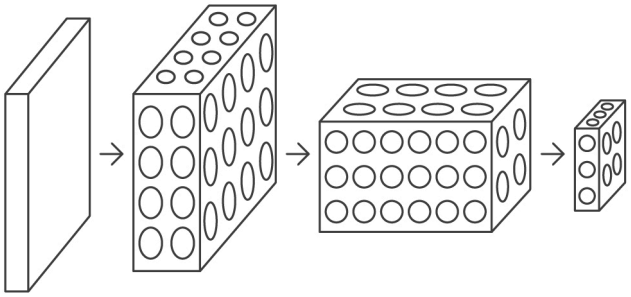

Figura 2.5: Red convolucional simple

Cuando hablamos de las redes neuronales en detección de rostros por ejemplo, hablamos sobre qué podría estar haciendo cada capa. Las primeras capas podrían estar detectando bordes y esquinas, las siguientes capas rasgos faciales, y la última una cara completa. En las RNC lo que utilizamos para detectar estas características son filtros. Un filtro puede detectar bordes verticales u horizontales, detectar zonas con circulos, texturas, etc. Realmente podrían detectar cualquier cosa. Lo que hacemos es entrenar estos filtros para que el conjunto de todos ellos sean capaces de generar un output que nos permita clasificar la imagen. Estos filtros no dejan de ser matrices de números que recorren la imagen y generan una salida tras realizar operaciones con los valores de la imagen.

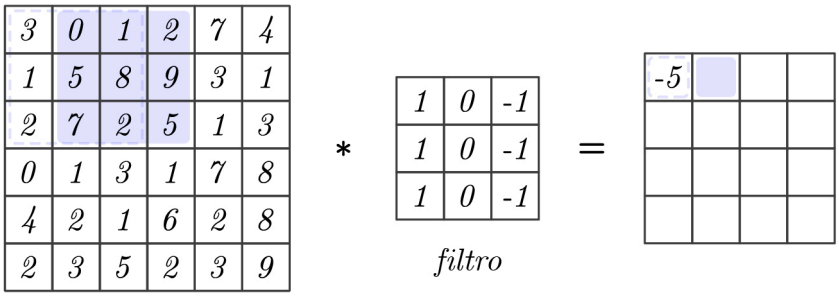

Figura 2.6: Capa convolucional con un filtro simple

Otro tipo de capas que se suelen utilizar son las denominadas *pooling layers*. Existen numerosos tipos de *pooling layers* con distintos propósitos y no tienen parámetros, es decir, no se entrenan. Es la persona que diseña la RNC la que tiene que seleccionar qué *pooling layer* va a colocar en cada capa y qué características va a tener. Lo que se va haciendo, al igual que en las capas convolucionales, es ir recorriendo la imagen en pequeñas áreas y realizar operaciones, pero esta vez utilizando solo los datos de la imagen. Estas capas tienen como objetivo principal reducir el

tamaño de los inputs y acelerar el cálculo. Dos de las *pooling layers* más utilizadas son *average pooling* y *max pooling*. La primera realiza la media de los valores de cada sección y la segunda selecciona el valor más alto.

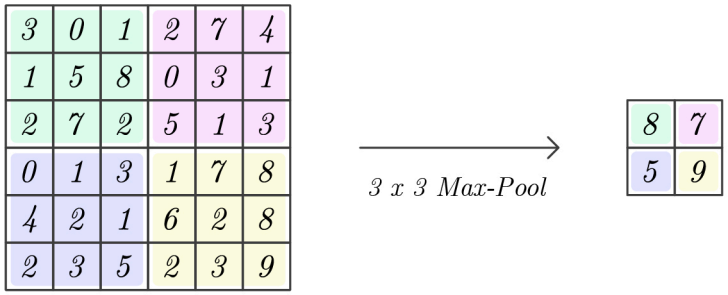

Figura 2.7: Capa con pooling layer

El uso de capas convolucionales y de *pooling layers* permite crear RNC muy potentes que llegan a clasificar imágenes en cuestión de segundos. Algunas de las redes convolucionales más conocidas son AlexNet (2), VGG (9) e Inception (11) que es la que hemos utilizado para la realización de este proyecto.

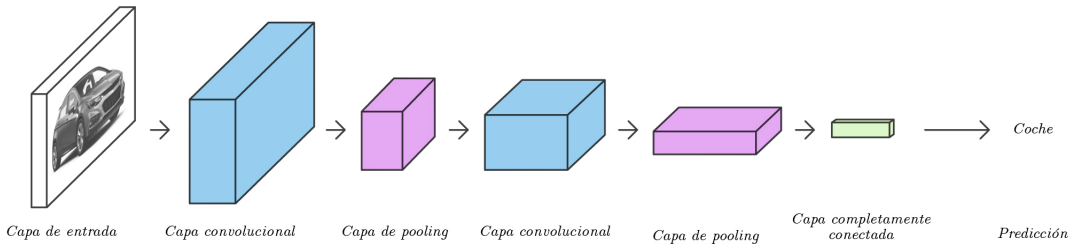

Figura 2.8: Red neuronal convolucional

2.2.2. Redes Neuronales Recurrentes

Las redes que hemos explicado hasta ahora trataban de clasificar un ejemplo dado (*input*) con una clase determinadas (*output*). Las RNC tenían la ventaja de clasificar imágenes mucho más rápido que las RN convencionales pero seguían teniendo la limitación de solo poder clasificar. A lo largo de la historia del DL se han ido desarrollando nuevas RN que han ido supliendo aquellas necesidades que iban surgiendo. Uno de estos avances son las RNR.

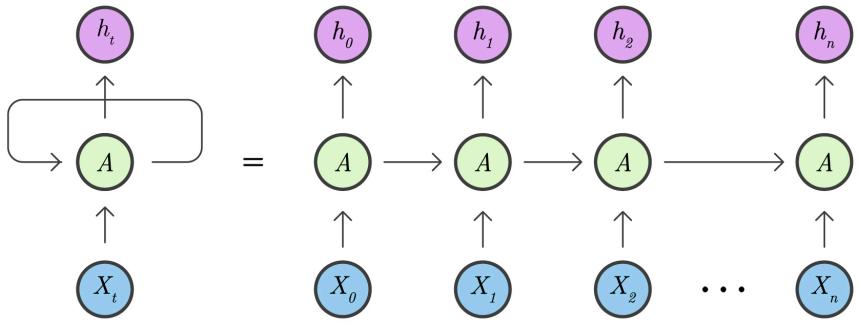

Figura 2.9: Red recurrente simple

Las RNR son un tipo de red que utilizan datos secuenciales tanto de *input* como de *output*. Estos datos pueden ser notas musicales, texto, audio, etc. Ya no necesitamos tener un valor de *input* o *output* fijo, si no que podemos generar cualquier tipo de información. Dependiendo de si el *input* o el *output* son una secuencia de datos o una única clase podemos encontrar distintas configuraciones de redes recurrentes[5]:

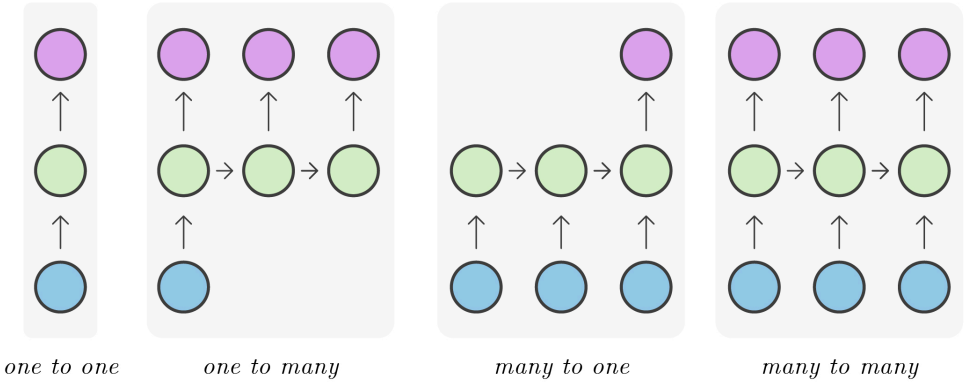

Figura 2.10: Tipos de redes recurrentes

- **one-to-one**: Las RN tradicionales. Por ejemplo, dada una imagen determinar si se trata de un perro o un gato.
- **one-to-many**: Red a la que dado un valor nos genera como *output* una secuencia. Por ejemplo, una RN que dada una primera nota sea capaz de generar una melodía en base a esa nota.

[5]Distintos tipos de redes recurrentes: https://builtin.com/data-science/recurrent-neural-networks-and-lstm

- **many-to-one**: Red a la que dada una secuencia la clasifica. Por ejemplo, identificar la voz de una persona con su propietario.
- **many-to-many**: Red a la que tanto el *input* como el *output* son secuencias. Por ejemplo, la traducción de texto.

Una de las características más interesantes de estas RN es su capacidad de mantener el contexto y recordar elementos pasados. Al ser alimentadas con el *output* de las capas anteriores, son capaces de utilizar esta información para generar el siguiente elemento de la secuencia[6]. Por ejemplo, en la generación de un texto es capaz de mantener el contexto del género y el número en una misma oración. No obstante si la secuencia se hace demasiado larga estas características comienzan a desvanecerse y se empieza a perder el contexto. Es aquí donde entran las redes *Long Short Term Memory* (LSTM).

2.2.3. Celdas LSTM

Las LSTM fueron introducidos por Hochreiter & Schmidhuber en 1997[7] y mejoradas y popularizadas por muchas personas a lo largo de los años. Son un tipo de celdas que proporcionan a la red la capacidad de recordar conceptos y elementos pasados con el fin de mantener un contexto. Son capaces de relacionar estos conceptos y de utilizarlos de forma más inteligente en la generación de la secuencia[8].

[6] Redes neuronales recurrentes: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

[7] Desarrollo de las redes LSTM: https://en.wikipedia.org/wiki/Long$_s hort - term_m emory$

[8] Información sobre LSTM: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

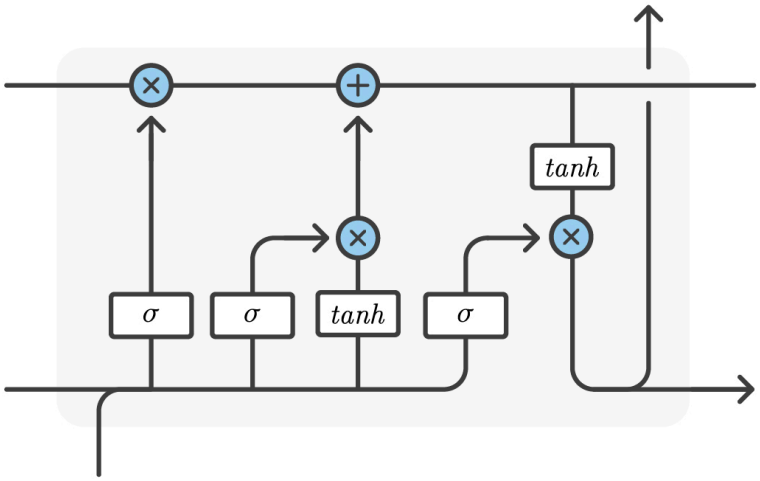

Figura 2.11: Celda LSTM

La estructura de estas redes es algo más compleja que la anterior ya que estas a parte de obtener la entrada y salida de la anterior celda son capaces de decidir con qué información se quedan y qué información eliminan, al igual que decidir qué información pasa a la siguiente celda.

2.2.4. Estructura Encoder-Decoder

Una vez familiarizados con las RNR podemos explorar más a fondo la arquitectura que nos interesa para este proyecto, la estructura *encoder-decoder*(10). Esta estructura se compone de dos partes con funcionalidades muy concretas:

- **Codificador (*encoder*)**: Tiene la función de encapsular el significado de los datos de entrada en un vector de características. En él quedarán mapeados los datos de tal forma que para entradas similares se obtengan vectores similares.
- **Decodificador (*decoder*)**: Partiendo de esta matriz de características el decodificador se encargará tanto de producir la salida esperada así como de comenzar la propagación hacia atrás del error. Está formado por un conjunto de unidades recurrentes conectadas las cuales producen una sección o token de la secuencia de salida y envían su estado a la siguiente unidad. Como cada unidad recibe el estado de la anterior puede generar su token basándose en el contexto de la secuencia hasta ese instante. Esto permite mantener una consonacia en los distintos tokens, como la concordancia de género y número en un texto.

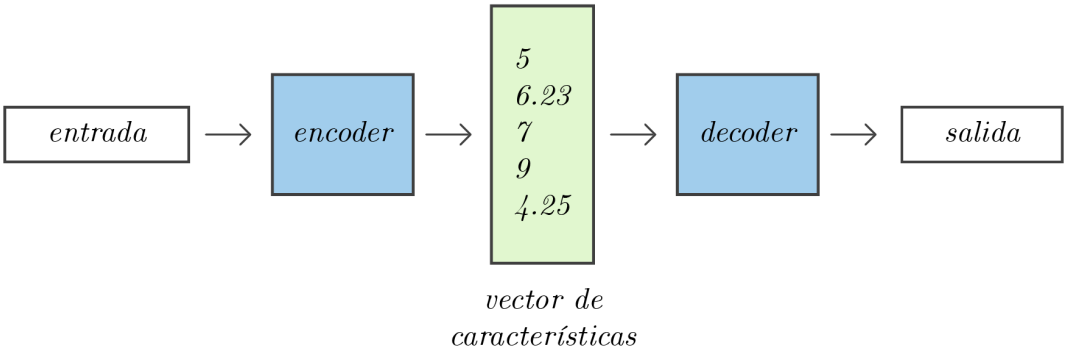

Figura 2.12: Estructura *encoder-decoder*

El codificador y el decodificador se pueden diseñar de distintas formas según lo requieran los tipos de datos de la entrada y de la salida. Además, cada una de las unidades del decoder puede generar valores nulos dando libertad a la red para elegir el tamaño ideal de la secuencia de salida.[9]

2.3. Word Embeddings

Las RN han supuesto un gran avance en el Procesamiento del Lenguaje Natural (PLN). Gran parte del significado de un lenguaje reside en las palabras de su vocabulario y en la relaciones semánticas de estas. *Word embeddings* (5) es una manera útil de almacenar el vocabulario de tal forma que un algoritmo pueda reconocer la semántica del mismo.

Una forma simplificada de representar un vocabulario conformado por n palabras sería asignar a cada palabra un índice i. Ahora podríamos representar a cada palabra del vocabulario con un vector de n bits en el que todos los bits son 0 excepto el que coincide con el índice de la palabra que queremos representar, que será 1. A este método se le conoce como representación one-hot y aunque sea útil para ordenar el vocabulario, no guarda ningún tipo de información sobre las relaciones de significado.

[9]Información sobre encoder-decoder: https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346

$$\begin{array}{cccc} \text{Rey} & \text{Reina} & \text{Manzana} & \text{Naranja} \\ (4914) & (7157) & (456) & (6257) \\ \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \end{array}$$

Para marcar las relaciones semánticas y sintácticas de las palabras dejaremos de representarlas con un bit a uno sino que lo haremos con un vector de números reales. Cada uno de estos números indicará el grado de relación que guarda una palabra con un concepto. De esta forma las palabras dejan de estar representadas con un vector para pasar a ser una matriz.

	Hombre	Mujer	Rey	Reina	Manzana	Naranja
Género	-1.00	1.00	-0.95	0.97	0.00	0.01
Real	0.01	0.02	0.93	0.95	-0.01	0.00
Edad	0.03	0.02	0.70	0.96	0.03	-0.02
Comida	0.09	0.01	0.02	0.01	0.95	0.97

Los valores reales que relacionan las palabras del vocabulario con los conceptos se pueden afinar con técnicas de deep learning de forma similar a como se ajustan los pesos de una red neuronal mediante el algoritmo del descenso del gradiente.. También se pueden usar valores preentrenados y probados como Word2vec (4) o *Glove* (7).

Partiendo de esta información una RNC puede construir frases con mejor estructura sintáctica y más sentido. Como en nuestro proyecto la intención es que la red genere oraciones con forma de pregunta, la capa de *Word embeddings* será de utilidad para ayudar a la red a usar correctamente los determinantes y pronombres interrogativos (qué, quién...). Además también será beneficioso para que el resto de la pregunta mantenga una estructura sintáctica correcta.

2.4. Visual Question Generation (VQG)

VQG(3) es una rama del *deep learning* que investiga cómo conseguir que un algoritmo sea capaz de generar preguntas relacionadas con imágenes. Este problema se

puede dividir en dos subproblemas, reconocimiento de imágenes y generación de preguntas. Es un problema muy similar al problema de descripción de imágenes (*image captioning*) con la diferencia de que ahora el texto generado de cada imagen es una pregunta.

Figura 2.13: Ejemplo de extracción de preguntas de imágenes

Esto se consigue aplicando conjuntamente algunas de las tecnologías previamente mencionadas. Mediante redes convolucionales para el reconocimiento de imágenes podríamos mapear la entrada en una matriz de características. Esta matriz sería la entrada de una RNR diseñada y entrenada para generar la pregunta. Aplicando así la arquitectura *encoder-decoder* seríamos capaces de aproximar el problema y, entrenando lo suficiente, obtener buenos resultados.

2.5. Idea Base

Al investigar sobre otros trabajos relacionados con una temática similar dimos con el trabajo realizado por Mariona Caros llamado *Sistema de generación de diálogos para terapia de reminiscencia* (8). En él elabora un *chatbot* que genera preguntas a partir de imágenes y ofrece un *feedback* a las respuestas que da el usuario.

Figura 2.14: Esquema de funcionamiento del algoritmo de Mariona Carós

El software se compone por dos funcionalidades principales. La primera consiste en un modelo VQG implementado en Pytorch. A través de este modelo genera las preguntas relacionadas con las imágenes suministradas por el usuario. La segunda parte es un modelo *Sequence to Sequence* que procesa las respuestas del usuario para generar un *feedback* más preciso que aporte cohesión a la conversación (1).

Cuando tuvimos acceso al código de Mariona nos dimos cuenta de que algunas de las funciones estaban desactualizadas, lo que afectaba negativamente al funcionamiento del *bot*. Esto, junto con otros problemas menores que parecían no tener fácil solución, nos hizo inclinarnos por implementar un nuevo *bot* desde cero. Creando nuestro propio *bot* podríamos utilizar otras tecnologías con las que estuviésemos más familiarizados y añadir nuevas funcionalidades.

Capítulo 3

Herramientas de desarrollo, tecnologías y metodología de trabajo

En este capítulo vamos a enumerar y explicar las diferentes aplicaciones, herramientas y tecnologías que hemos utilizado para trabajar en el proyecto, ya sea para el desarrollo software, planificación de tareas o comunicarnos entre nosotros. También detallaremos la forma en la que todos los miembros del equipo trabajamos con dichas herramientas y cómo hemos organizado nuestro trabajo.

3.1. Herramientas de desarrollo

Durante los primeros días de reuniones nos pusimos de acuerdo sobre las aplicaciones y herramientas que íbamos a utilizar a la hora de desarrollar nuestro trabajo. Estas herramientas y aplicaciones mejoraron muestra experiencia a la hora de programar, comunicarnos y organizarnos. A continuación vamos a explicar de forma breve las aplicaciones con las que hemos trabajado:

- **Visual Studio Code**: Se trata de un entorno de programación de código abierto muy utilizado para programar aplicaciones sencillas como la nuestra. Nos permitió programar en Python de forma muy cómoda y visual ya que integra este lenguaje de programación a la perfección. Presenta una gran cantidad de plugins que nos sirvieron de gran ayuda, como por ejemplo **Live Share**, que permite reproducir la técnica de *pair programming* a distancia y así conseguir los tres miembros del grupo programar a la vez en tiempo real, lo cual agilizó nuestro trabajo.

- **Github**: Es un repositorio online que nos permite subir nuestra aplicación y mantener un historial con los cambios realizados en el proyecto por cada uno de los miembros del equipo.
- **GitKraken**: Con GitKraken controlamos todo lo relacionado con GitHub y el control de versiones. Permite realizar acciones como *pull*, *push* o *commits* de una forma más intuitiva ya que no es necesario el uso de comandos para realizar estas acciones. Permite además poder acceder al código de versiones anteriores y crear nuevas ramas, las cuales se muestran de forma visual en forma de árbol.
- **Trello**: Se trata de una herramienta online que utilizamos para definir y organizar objetivos, tareas, reuniones e ideas, las cuales se pueden asignar a diferentes miembros del equipo e indicar la fase en la que se encuentran (en proceso, pruebas, terminadas, etc.). De esta forma, mantuvimos una buena organización de nuestro proyecto en todo momento, lo que mejoró nuestro rendimiento.
- **Discord y Google Meet**: Utilizamos Discord para reunirnos virtualmente con el fin trabajar de forma conjunta programando o desarrollando el proyecto en general. Las reuniones con nuestro tutor bisemanales las realizábamos usando Google Meet.
- **Telegram**: No solo utilizamos Telegram para comunicarnos entre nosotros de forma rápida y cómoda sino que fue la herramienta que decidimos utilizar para implementar nuestra aplicación debido a su facilidad para crear *bots*.
- **Google Drive**: Utilizamos Google Drive para diferentes fines. Primeramente como gestor de archivos online. Ahí subíamos resúmenes de las reuniones que hacíamos, documentación del código que íbamos desarrollando, copias de seguridad, etc. También lo utilizamos para poder entrenar nuestra red debido a la potencia de su herramienta Google Colab que da la posibilidad de ejecutar código en Python mediante *notebooks*.
- **Anaconda Navigator**: Es un ambiente de trabajo creado para ciencia de datos que permite hacer funcionar diferentes aplicaciones y administrar fácilmente distintos paquetes en Python. Gracias a esta herramienta pudimos crear un *environment* común e independiente para la realización de nuestro proyecto.

3.2. Tecnologías

Después de buscar proyectos existentes que realizasen una tarea similar a lo que queríamos implementar decidimos pensar otra forma de hacer nuestra aplicación de manera más sencilla. Para ello comenzamos buscando tutoriales e información sobre

diferentes bibliotecas de código abierto en Python encontrando finalmente Keras, que era perfecta para lo que queríamos hacer.

3.2.1. Python

Todo el código de nuestro proyecto ha sido programado en Python debido a las grandes ventajas que este lenguaje de programación de alto nivel tiene. Python es un lenguaje muy versátil que también destaca por su legibilidad y limpieza, ya que la propia sintaxis del código te obliga a mantenerlo ordenado y bien estructurado. Además, cuenta con un gran número de bibliotecas y multitud de recursos de código abierto disponibles online. Este lenguaje es también muy elegido a la hora de programar aplicaciones de IA por la existencia de librerías muy potentes de ML como Tensorflow, Keras, etc.

Todo esto hizo que nos decantásemos finalmente por elegir este lenguaje como base de nuestro proyecto.

3.2.2. Keras

Keras es una biblioteca de código abierto programada en Python. El objetivo de esta biblioteca es aumentar la eficiencia y rapidez a la hora de crear RN. Para ello, en lugar de funcionar como un *framework* independiente, Keras funciona como una *Application Programming Interface* (API) que permite acceder a varios *frameworks* de ML y desarrollarlos. Entre los *frameworks* compatibles con Keras se encuentra por ejemplo TensorFlow[1].

Keras utiliza las bibliotecas de los *frameworks* de ML vinculadas (que en cierto modo actúan como un motor de *backend* para Keras) en lugar de ser utilizado para realizar operaciones sencillas de bajo nivel. Esta es una clara diferencia que tiene respecto a un *framework* normal. Además, el usuario de esta biblioteca no necesita entender ni controlar directamente el propio funcionamiento del *backend* del *framework*, ya que las diferentes capas de la RN a configurar se relacionan entre sí de acuerdo con el principio modular. De esta forma, las acciones necesarias por parte de los usuarios se reducen al mínimo.

Keras ha sido muy importante para nosotros ya que a la hora de programar nuestro proyecto hemos utilizado un gran número de funciones predefinidas en esta biblioteca, lo que nos ha ayudado a simplificar y agilizar el proceso de creación de la RN gracias a su usabilidad y a todas las características mencionadas anteriormente.

[1]Información sobre Keras: https://www.ionos.es/digitalguide/online-marketing/marketing-para-motores-de-busqueda/que-es-keras/

3.2.3. Environment

Para evitar posibles problemas y conflictos con Python y sus librerías y paquetes decidimos crear un entorno o *environment* común, donde todos los integrantes del grupo tuviésemos instaladas las mismas versiones de los paquetes a utilizar. Para ello utilizamos Anaconda, con una versión de Python 3.7.7 e instalamos los siguientes paquetes:

- **Python Telegram Bot**: Proporciona toda la funcionalidad para poder realizar la interacción con el *bot* de Telegram.
- **Pillow**: Nos permite trabajar cómodamente con imágenes
- **Tensorflow**: Librería de *data science* para poder realizar de forma sencilla RN.
- **Google Trans New**: Librería que nos permite traducir un texto a cualquier idioma utilizando el traductor de Google.
- **Inflect**: Nos permite trabajar con texto a la hora de transformarlo.
- **FPDF**: Nos permite generar un *Portable Document Format* (PDF) mediante código.

3.3. Metodología de trabajo

La metodología empleada a la hora de realizar nuestro proyecto se ha basado en el modelo Kanban, perteneciente a las Metodologías Ágiles que son empleadas en el desarrollo software. Kanban es una herramienta para mapear y controlar el flujo de trabajo. Consiste en colocar tableros con distintos estados de las tareas e ir moviendo estas según vayan avanzando en el proceso de desarrollo.

Esta decisión se debe principalmente a que somos un grupo de trabajo pequeño, formado únicamente por tres estudiantes, y a la necesidad de modificación de tareas, requisitos y código a medida que se iba avanzando durante el trabajo. La elección de este modelo de metodología ágil nos permite adaptarnos con más libertad y flexibilidad al ciclo de trabajo.

Siguiendo la estrategia Kanban y gracias a la herramienta Trello explicada anteriormente, realizamos un diagrama de tareas en el que se reflejan tres columnas: pendientes, en proceso o terminadas. Este cuadro está al alcance de todos los miembros del equipo, evitando así la repetición de tareas o la posibilidad de que se olvide alguna de ellas.

Además, todos los miembros del equipo hemos estado en contacto continuamente gracias a aplicaciones de mensajería instantánea como Telegram, donde nos avisábamos sobre posibles cambios de hora de reuniones, tareas completadas y coordinación para el entrenamiento de la red entre otras cosas.

A su vez, mantuvimos reuniones online cada dos semanas a través de Google Meet con nuestro tutor, donde realizábamos un seguimiento de nuestro trabajo hasta la fecha y resolvíamos dudas o problemas que iban apareciendo durante el desarrollo del proyecto.

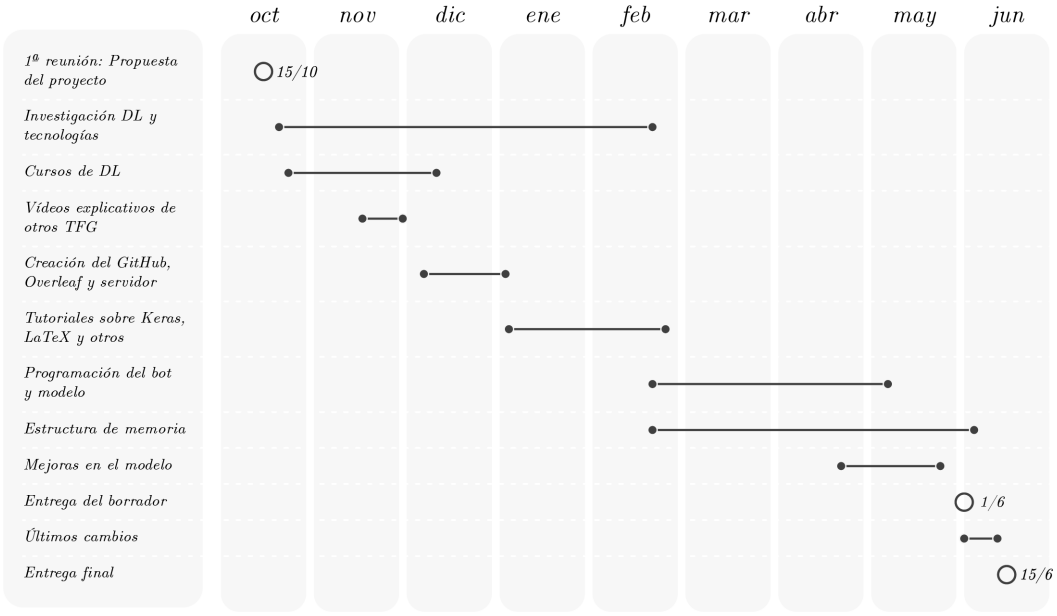

Figura 3.1: Planificación de tareas durante el proyecto

En la figura anterior podemos observar el reparto de tareas e hitos que hemos seguido todos los integrantes del grupo durante el desarrollo de nuestro trabajo:

- **1ª Reunión y propuesta del proyecto**: El día quince de octubre tuvimos la primera reunión grupal entre los tres miembros del grupo de trabajo y el tutor. En esta reunión definimos las principales ideas para la realización del proyecto y se resolvieron dudas en torno a este.
- **Investigación DL y tecnologías**: Durante esta fase realizamos todo tipo de tareas relacionadas con el comienzo y la preparación del proyecto además del refuerzo de conceptos sobre DL.
    - **Cursos DL**: Utilizamos unos cursos de DL explicados anteriormente en este documento para ampliar y reforzar nuestros conocimientos.

  - **Vídeo explicativo para otros TFGs**: Durante las últimas semanas de Noviembre, tuvimos que realizar una pequeña exposición en forma de vídeo explicando las ideas y objetivos de nuestro trabajo. Esta exposición se envió a otros grupos de alumnos que estaban realizando proyectos relacionados con terapias para personas con problemas de memoria.
  - **Creación de Github y Overleaf común, acceso al servidor**: Para organizar el proyecto, utilizamos varias aplicaciones y herramientas, creando en Google Drive documentos compartidos, un repositorio de Github común, un documento de Overleaf para la escritura de la memoria, etc. Todo esto completamente accesible para todos los miembros del grupo y el tutor.
  - **Tutoriales Keras, Latex y otras tecnologías**: Ya con el proyecto organizado y con los conocimientos de DL completamente reforzados el último paso que dimos antes de comenzar a programar nuestros modelos fue el de realizar búsquedas y tutoriales sobre las nuevas tecnologías que íbamos a tener que utilizar para realizar nuestro trabajo como Keras o Latex.

- **Programación de los modelos y bot de Telegram**: Gracias a todo el trabajo realizado antes de comenzar esta tarea, pudimos realizar la programación de nuestra RN y nuestro bot de una forma mas ágil.
- **Escritura de la memoria**: La escritura de la memoria se llevó a cabo al mismo tiempo que íbamos avanzando en la programación de nuestros modelos.
- **Mejoras en los modelos**: Realizamos diferentes mejoras en los modelos del bot y de la red neuronal, incluyendo Glove Embeddings, nuevos mensajes y preguntas y ampliando el conjunto de datos de entrenamiento.
- **Entrega de borrador**: Este hito sirvió para que nuestro tutor diese el visto bueno a lo que hasta el momento llevábamos trabajado y escrito en la memoria.
- **Completar y corregir memoria**: Realizamos correcciones en el contenido de la memoria como por ejemplo faltas de ortografía y añadimos nuevo contenido en ella.
- **Entrega final**: Este hito del día quince de junio ponía fin a nuestro trabajo. Entregamos nuestro código y memoria finalizados.

Capítulo 4

Remi - Bot para terapia de reminiscencia basado en VQG

Nuestro proyecto se divide en dos módulos que se relacionan entre sí para poder simular la terapia de reminiscencia, el *bot* de Telegram y una RN programada en Keras para la generación de preguntas visuales VQG a partir de imágenes. Utilizamos Telegram como interfaz para poder interactuar con el usuario. Además de esto, realizamos dos aplicaciones para la generación y formateo de los datos que se ejecutan de forma independiente en consola.

La tarea de VQG implica generar preguntas significativas basadas en la imagen de entrada. Es un problema multimodal que involucra la comprensión de imágenes y la generación de lenguaje natural, especialmente utilizando métodos de aprendizaje profundo.

Esta tarea es significativamente más difícil en comparación con las tareas de clasificación de imágenes o reconocimiento de objetos que han sido bien investigadas.

El *bot* de Telegram interactuará con el paciente, trasladando las preguntas generadas por la RN al usuario para que este pueda responderlas, simulando así una sesión de terapia de reminiscencia. Además cuenta con otro tipo de funciones y permitirá al usuario cambiar de pregunta o foto, subir imágenes para utilizar en la terapia, eliminar datos personales, etc.

4.1. Obtención de conjuntos de datos

Una de los puntos más importantes para la realización de nuestro proyecto era conseguir un gran volumen de datos relacionados con las técnicas utilizadas para la generación de preguntas visuales. Estos conjuntos de datos presentan tanto imágenes como preguntas asociadas a estas.

El principal objetivo en esta fase del proyecto era encontrar el mayor volumen de datos posibles para poder entrenar nuestro modelo VQG una vez estuviese completamente terminado. Cuanto mejor sea la calidad de los datos obtenidos y mayor sea el entrenamiento del modelo, mejores serán los resultados y preguntas obtenidas.

Para ello realizamos una búsqueda intensiva en diferentes páginas de internet optando al final por utilizar estos conjuntos de datos:

- Dataset de Microsoft Coco[1].
- Dataset de Flickr, Bing y Coco[2].

En la mayoría de los conjuntos de datos que recopilamos, cada imagen está asociada a cinco preguntas diferentes sobre los elementos presentes en la fotografía. Al asociar cada imagen con varias preguntas, el *dataset* captura parte de la variedad lingüística que se puede utilizar para describir la misma.

Gran parte de las páginas y links en las que realizamos la búsqueda tenían los datos en inglés, lo que suponía un nuevo reto para nosotros a la hora de desarrollar el proyecto. Decidimos que nuestro modelo VQG trabajase con preguntas en inglés y después crear un módulo de traducción de preguntas (el cual explicaremos mas adelante en la memoria) para que el *bot* de Telegram y la interacción con los pacientes se pudiese realizar en diferentes idiomas.

4.2. Módulo de procesado de datos

Tras buscar sin mucho éxito datasets con preguntas e imágenes relacionadas con el campo de la terapia ocupacional nos vimos obligados a buscar otros datasets más genéricos. Como nuestra intención es extraer preguntas genéricas de imágenes sobre lo que hay en la foto nos podría valer con cualquier dataset que cumpliese este requisito.

[1]Dataset de Microsoft Coco: https://visualqa.org/download.html

[2]Dataset de Flickr, Bing y Coco: https://www.microsoft.com/en-us/download/details.aspx?id=53670

No obstante estos, al ser de fuentes diferentes, teníamos que reestructurarlos y procesarlos para obtener un formato común y así poder utilizar cualquiera de ellos indistintamente.

4.2.1. Módulo de dataset

Al ser tantos datos, necesitábamos automatizar de alguna forma la generalización de todos ellos. En consecuencia, desarrollamos un módulo aparte llamado Datasets que nos permitió formatear de igual forma todos los *datasets* para poder utilizar cada uno indistintamente. La ejecución de este módulo es en consola e independiente de la ejecución principal del programa ya que no está pensada para un uso por el usuario sino por el desarrollador que implemente nuestro sistema. Aunque los datos nos venían de varias fuentes diferentes teníamos dos tipos principales:

- Flickr, Bing y coco: Estos *datasets* estaban formados por un archivo csv que contenía un enlace a la página web donde estaba almacenada la pregunta y varios datos más en los que se incluían las preguntas.
- MsCoco: Este *dataset*, que es el más grande, ya tenía por un lado las imágenes descargadas (había que descargarlas previamente mediante un zip) y en otro fichero tenía las preguntas identificadas por el id de la imagen.

Una vez descargados estos *datasets* los colocamos en el directorio source/datasets/data_to_download, lugar donde el programa los buscará para descargarlos y formatearlos.

Al ejecutar el programa aparece un menú indicando al usuario las posibles acciones que puede tomar:

```
Choose the action to take:
   1. Download datasets.
   2. Load datasets.
   3. Format mscoco database.

Choice:
```

Como vemos la primera opción permite descargar los *datasets* que faltan, la segunda opción permite cargarlos (está opción no tiene utilidad salvo la de confirmar que el *dataset* que se cargue está correcto) y la tercera opción permite formatear el *dataset* de mscoco.

Si elegimos la primera opción nos preguntará si queremos descargar todo el dataset o únicamente un dataset en específico.

```
Choose the action to take:
   1. Download all datasets.
   2. Download a specific dataset.

Choice:
```

En caso de elegir un dataset específico preguntará por el nombre de este y el conjunto de datos a descargar (train, test o val) que hacen referencia al conjunto de entrenamiento, prueba y validación. En nuestro caso, pese a que hacemos la distinción utilizamos los tres como un único conjunto ya que no realizamos pruebas de validación por ser una tarea compleja al tratarse de una red con salidas de diferentes longitudes y se escapa de los objetivos de este proyecto. Obtendremos estos mensajes:

```
Enter the name of the dataset to download (coco, flickr, bing):
Enter the name of the set to download (train, validation, test):
```

A medida que se van descargando vamos mostrando por consola el progreso y cada imagen descargada. Al estar las imágenes alojadas en sitios web es normal que haya muchos que no estén ya disponibles y al intentar descargarla de error. En este caso lo que hacemos es descartar la imagen y las preguntas y mostrar al usuario por consola que esa imagen no se ha podido descargar. En una descarga normal obtendremos una salida por consola parecida a esta:

```
[ Downloading images from the dataset flickr_train ]
[DONE] Image 1/2449 with id 3432234 correctly downloaded.
[DONE] Image 2/2449 with id 6853466 correctly downloaded.
[ERROR] Image 3/2449 with id 3224635 is not available to downaload.
```

Una vez descargados los *datasets* se habrá generado una carpeta llamada data con los *datasets* y las preguntas formateadas. Para comprobar que estos datos se han cargado correctamente hemos desarrollado la opción 2 del menú que permite comprobarlo.

```
Enter the name of the dataset to load (coco, flickr, bing, mscoco):
    flickr
Enter the name of the set to load (train, validation, test, all):
    train
The dataset has been successfully loaded.
```

Por ejemplo, una vez descargado al probar el conjunto de datos *train* del *dataset* Flickr el programa nos indica que se ha cargado correctamente.

Finalmente tenemos la tercera opción que formateará las imágenes y las preguntas. En este caso como no hay que descargar nada ya que tenemos que descargarlas con anterioridad el proceso es mucho más rápido. Al igual que cuando descargamos, esta opción irá mostrando por consola el proceso de formateo de cada imagen:

```
[DONE] Image 0/82.783 with id 3242354 correctly saved.
[DONE] Image 1/82.783 with id 4325231 correctly saved.
[DONE] Image 2/82.783 with id 9098112 correctly saved.
```

Una vez realizados estos pasos tendremos todos los datos correctamente formateados. A la hora de nombrar las imágenes decidimos dejar el mismo id que venía ya por defecto para evitar tener que comprobar repetidos. Para poder distinguir entre una imagen de un *dataset* y otra de otro en caso de que ambas tuviesen el mismo identificador decidimos incluir delante del id el nombre del *dataset*. Es decir, por ejemplo la imagen con id 3245124 del *dataset* Bing tendrá un nuevo id bing_3245124. Utilizamos este nuevo id como nombre de la imagen y como identificador de las preguntas.

4.2.2. Módulo de Vocabulary

Después de haber recopilado y preparado todos los conjuntos de datos, el siguiente paso es limpiar los datos y generar un vocabulario compuesto por las palabras más comunes y repetidas en las preguntas que componen los datos de entrenamiento. Al igual que con el *dataset*, creamos un módulo de ejecución aparte del módulo principal que se accede mediante la consola. Este módulo a diferencia del anterior no posee una interfaz donde seleccionar la tarea a ejecutar ya que hace una única función, preparar el vocabulario. Para nuestro proyecto generamos dos vocabularios diferentes:

- **Vocabulario completo**: Este vocabulario es un diccionario compuesto por todas las palabras únicas presentes en todas las preguntas correspondientes a

la imágenes. A su vez, cada término estará compuesto por un identificador y un contador el cual indicará el numero total de veces que se repite dicha palabra. Además, el diccionario contará con la longitud (en palabras) de la pregunta más larga del conjunto de todos los datos.

- **Vocabulario reducido**: Este diccionario solo contendrá un numero reducido de palabras (las más repetidas) ya que obviamos aquellas que tienen muy pocas repeticiones para evitar introducir mucho ruido o faltas de ortografía. Mantendrá también la longitud de la pregunta más larga, ya que se utilizará más adelante. También incluirá la cuenta total de las palabras que componen el vocabulario.

El primer paso para poder crear nuestro vocabulario completo es obtener las palabras que formaban cada una de las preguntas pertenecientes a los datos recogidos. Para ello es necesario una limpieza básica del texto para deshacernos de signos de puntuación y convertir nuestras palabras a minúsculas.

Después de esto, solo queda controlar si la palabra ya forma parte del diccionario, si es así, se aumenta su contador de apariciones y en caso contrario se incluye la nueva palabra.

Generar nuestro vocabulario reducido es mucho mas fácil. Establecemos como variable global un *threshold* o umbral, el cual creímos que era conveniente mantenerlo en diez unidades ya que de esta forma omitimos aquellas palabras mal escritas o con faltas de ortografía que aparecen en muy pocas ocasiones. Esta decisión se toma en base a la necesidad de obtener un vocabulario expresivo y lo más pequeño posible.

Cabe destacar que el vocabulario reducido contendrá cuatro tokens: <start>, <end>, <pad>y <unk>. Estos tokens nos ayudarán a conocer donde empieza y donde acaba la pregunta generada, así como si presenta un padding, añadiendo el token <pad>hasta llegar a la longitud de pregunta máxima. Las palabras desconocidas se asignan a un token <unk>durante el entrenamiento, pero no permitimos que el decodificador produzca este token en el momento de la prueba.

La generación de dos diccionarios diferentes, el completo y luego el reducido, se debe también a lo explicado anteriormente, la necesidad de obtener un vocabulario expresivo y lo mas pequeño posible. De esta forma primero generamos el vocabulario con todas las palabras diferentes y después filtramos para obtener un vocabulario más pequeño.

El uso del vocabulario es imprescindible para que durante la predicción Beam Search (explicada mas adelante) se puedan proporcionar probabilidades a los tokens o palabras que forman dicho vocabulario con el objetivo de generar las preguntas más probables.

Inmediatamente después de producir nuestro vocabulario reducido, creamos dos diccionarios extras con el objetivo de realizar un mapeo y asignar las palabras a un índice y viceversa.

4.3. Generación de preguntas - Modelo VQG

Abordaremos la tarea de VQG creando una estructura de *encoder-decoder*. Nuestro modelo del Encoder combinará tanto la forma codificada de la imagen como la forma codificada de las preguntas correspondientes y pasará estas características al modelo del Decoder.

Nuestro modelo VQG presentará tres pasos principales:

1. Procesado de la secuencia de texto (pregunta).
2. Extracción del vector de características de la imagen.
3. Decodificación de las salidas utilizando para concatenar las dos capas anteriores.

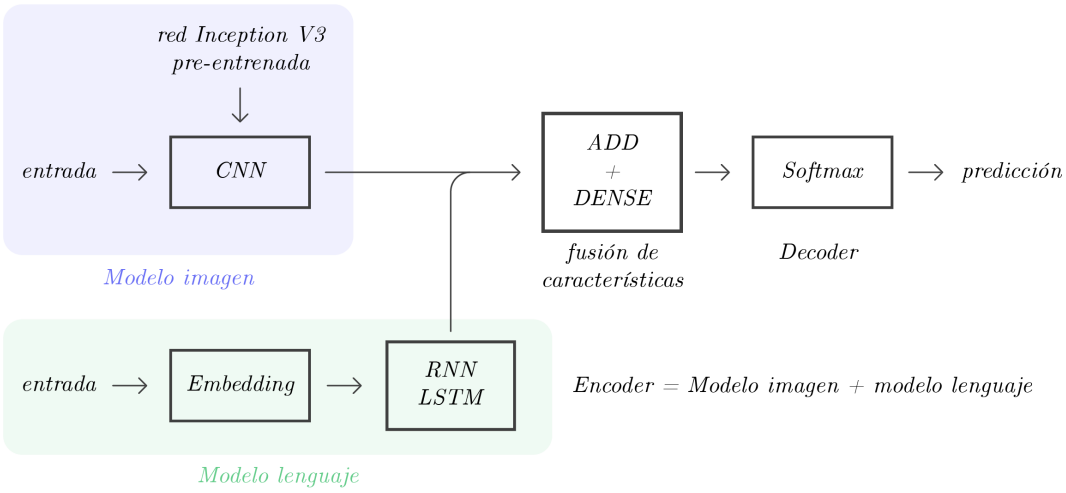

Figura 4.1: Diagrama del modelo

Como se puede observar en el diagrama, dividimos nuestro Encoder en dos submódulos llamados modelo de imagen y modelo de lenguaje. El modelo de imagen utilizará la tecnología de las RNC mientras que para el modelo de lenguaje optamos por las RNR.

El Decoder recoge las salidas previamente fusionadas de los módulos que componen el Encoder y las procesa para poder realizar una predicción final.

4.3.1. Encoder

Tal y como hemos mencionado anteriormente, nuestro Encoder estará formado por dos módulos bien diferenciados, el modelo de imagen y el modelo de lenguaje.

Creamos estos dos módulos por separado para mantener la imagen fuera de la RNR y así poder entrenar la parte de la RN que maneja imágenes y la parte que maneja el lenguaje por separado, usando imágenes y oraciones de conjuntos de entrenamiento separados.

Para la creación del modelo de imagen y para realizar la codificación de las características de nuestras imágenes hemos optado por transferir el aprendizaje utilizando la red *InceptionV3*[3], que está preentrenada en el conjunto de datos ImageNet[4].

InveptionV3 es una red neuronal *open-source* desarrollada por Google. Decidimos usarla ya que permite clasificar imágenes en 1.000 clases diferentes y con el menor número de parámetros de entrenamiento en comparación con las otras opciones como ResNet o VGG-16. A nosotros no nos interesa clasificar estas imágenes, sin embargo, todo el aprendizaje que ya tiene esta red nos va a ayudar a obtener todas las características que queremos de las imágenes.

Estas características son las que le pasaremos al Decoder para que pueda hacer la generación de la pregunta. Por lo tanto, lo que hacemos es eliminar la capa *softmax* que es la capa encargada de clasificar.

[3]Página web de InceptionV3: https://keras.io/api/applications/inceptionv3/
[4]Página web de ImageNet: https://www.image-net.org

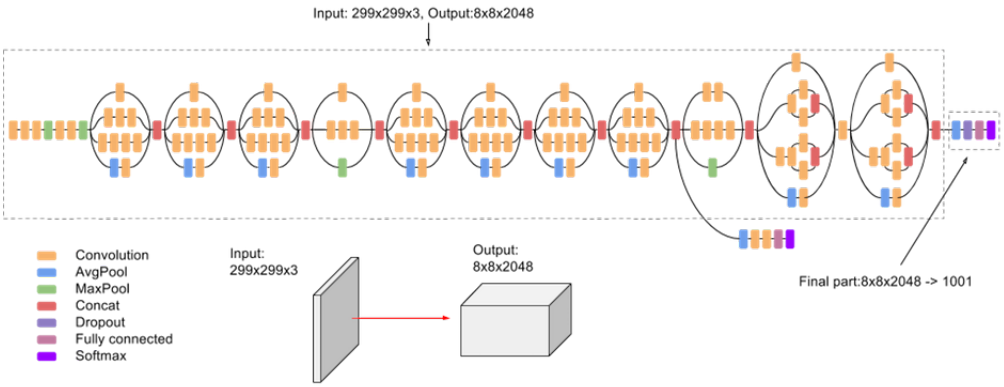

Figura 4.2: Diagrama del modelo

Debido a que estamos utilizando *InceptionV3* en nuestro modelo, necesitamos aplicar un pequeño preprocesado a nuestra imagen de entrada antes de introducirla en el modelo. Hemos definido una función de preproceso para remodelar las imágenes a una dimensión (299 x 299) para después pasarlas a través de una función definida por *Keras* denominada `preprocess_input()`.

Una vez tenemos la imagen preparada, ya podemos continuar y codificar nuestras imágenes, es decir, extraer los vectores con las características de la imagen de dimensión (2048, 0).

La variable `input_2` es el vector de imagen (características) extraído por nuestra red InceptionV3. Le sigue también una capa Dropout de *0.5* con el fin de evitar que se produzca sobreajuste. Finalmente pasa por una capa completamente conectada.

```
inputs2 = Input(shape=(2048,))
fe1 = Dropout(0.5)(inputs2)
fe2 = Dense(256, activation='relu')(fe1)
```

En cuanto al modelo de lenguaje, utilizaremos una capa de Embedding, además utilizaremos un dropout de 0,5 para poder evitar el fenómeno conocido como *overfitting* o ajuste excesivo. Finalmente una LSTM para realizar el procesado y codificado de las preguntas asociadas a las fotografías que se están codificando a su vez en el modelo de imagen.

```
inputs3 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, embedding_dim, mask_zero=True)(inputs3)
se2 = Dropout(0.5)(se1)
se3 = LSTM(256)(se2)
```

4.3.2. Decoder

La combinación de las características de la imagen con las codificaciones de texto en una etapa posterior y avanzada de la arquitectura es ventajosa y puede generar preguntas de mejor calidad con capas más pequeñas que la arquitectura de tradicional (RNC como codificador y RNR como decodificador).

Después de definir los modelos del Encoder, se utiliza una capa de adición (*Add*) para poder concatenar los vectores obtenidos de ambos modelos (imagen y lenguaje). Con la salida de esta capa, se alimenta otra capa totalmente conectada con activación *relu* (`dense_1`). La salida se obtiene a partir de una capa *softmax* (`dense_2`) que proporciona probabilidades a nuestro vocabulario.

```
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)

model = Model(inputs=[inputs2, inputs3], outputs=outputs)
```

En cuanto al entrenamiento del modelo, el primer paso que realizamos es compilarlo utilizando `categorical_crossentropy` como función de pérdida y *Adam* como optimizador.

```
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

La función de pérdida de `categorical_crossentropy` nos ayudará a calcular la pérdida de entropía cruzada entre las etiquetas (en nuestro caso las palabras de nuestro vocabulario) y las predicciones realizadas por el modelo [5]. Además, el optimizador Adam es un método de descenso de gradiente estocástico que se basa en la

[5]Información sobre la función CategoricalCrossentropy en Keras: https://keras.io/api/losses/probabilistic_losses/#categoricalcrossentropy-class

estimación adaptativa de momentos de primer y segundo orden. Es computacionalmente eficiente e invariante al reajuste diagonal de gradientes, tiene pocos requisitos de memoria y es muy adecuado para problemas grandes en términos de datos o parámetros[6].

Debido al gran volumen de datos que tenemos, lo siguiente es realizar una función con la que podamos entrenar los datos en pequeños conjuntos o *batches* [7]. Esta función, llamada `data_generator()` contará con un bucle sobre todas las imágenes. Cada imagen se codificará utilizando el modelo de imagen perteneciente al Encoder y para cada pregunta asociada a dicha imagen se realizán las siguientes acciones:

1. Codificar la pregunta utilizando el diccionario de mapeo `word-to-index` generado al crear el vocabulario.
2. División en un par entrada-salida.
3. Realizar un *padding* de la secuencia de entrada con la función `pad_sequence()`.
4. Codificar la secuencia de salida con la función `to_categorical()`.

Al finalizar todos estos pasos, se guardan toda la información codificada en tres diferentes listas, una para la foto y otras dos para las secuencias (preguntas) de entrada y salida. Justo después comprobamos si el número de iteraciones realizadas llegan al tamaño de *batch* definido como uno de los parámetros de la función y si se cumple este requisito creamos el *batch* de datos formados por las listas `X1`, `X2` e `Y` antes de volver a inicializar dichos *arrays*.

Finalmente, creamos una función llamada `decoder_training()` donde combinamos los dos pasos explicados anteriormente para realizar el entrenamiento del modelo durante quince *epochs* y un tamaño de *batch* igual a *32*. [8]

4.3.3. Predicción Beam Search

Para poder obtener las mejores preguntas, utilizamos la técnica de Beam Search (12) creando una nueva función `beam_search_predict()`. Esta función presenta un parámetro llamado `beam_size` al cual llamaremos k durante la explicación.

En el primer paso de tiempo, seleccionamos k *tokens* (palabras) con las probabilidades condicionales más altas, es decir, las k predicciones principales. Cada uno

[6]Información sobre el optimizador Adam en Keras: https://keras.io/api/optimizers/adam/

[7]Batch: división del conjunto completo de datos en subconjuntos denominados batches

[8]Epoch: Una época o *epoch* se completa cuando un conjunto de datos entero se pasa hacia adelante y hacia atrás a través de la RN una única vez.

de ellos será el primer token de k secuencias de salida candidatas, respectivamente. En cada paso de tiempo posterior, con base en las k secuencias de salida candidatas en el instante anterior, continuamos seleccionando k secuencias de salida candidatas con las probabilidades condicionales más altas.

Por lo tanto, la lista siempre contendrá las k predicciones principales. En nuestro caso, utilizamos un tamaño o `beam_size` igual a diez para poder obtener las diez preguntas con mayor probabilidad. De esta decena de preguntas, se mostrarán únicamente cuatro cuestiones de forma aleatoria al usuario (junto con la primera pregunta general, haciendo así un total de 5 preguntas), para que, en caso de que vuelva a repetir la terapia con la misma foto, pueda responder preguntas diferentes.

4.4. *Bot* de Telegram

Como ya hemos comentado anteriormente decidimos implementar nuestro proyecto mediante un *bot* de Telegram debido a todas las ventajas que esto nos proporciona:

1. Nos permite desarrollar de forma cómoda y fácil una interfaz con la que interactúa el usuario.
2. Al tratarse de una app de mensajería resulta muy familiar e intuitivo para cualquier persona utilizarlo.
3. Al tratarse de una aplicación tan reconocida y utilizada existen numerosos recursos online y tutoriales que nos facilitan la implementación.

Antes de comenzar con la programación del *bot* tuvimos que crearlo como tal. Para ello utilizamos BotFather que es el *bot* oficial de Telegram con el que se crean otros *bots*. Lo creamos utilizando el comando y dándole un nombre. Una vez creado BotFather nos indicará un código *token* que tendremos que introducir en nuestro programa.

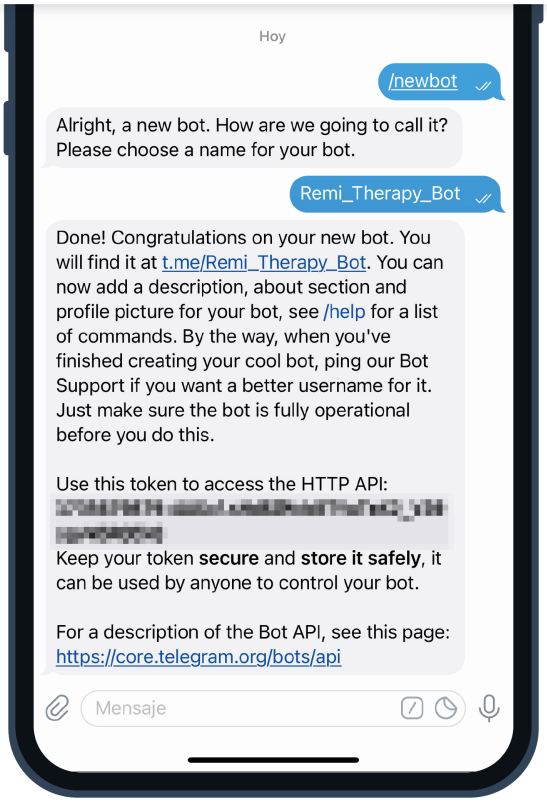

Figura 4.3: Conversación con BotFather para crear un bot

En el GitHub oficial de *bots* de Telegram en Python encontramos algunos ejemplos de bots ya creados para tomar de referencia. En nuestro caso al tratarse de un *bot* interactivo utilizamos como ejemplo `conversationbot2.py` que se trata de un *chatbot* con el que mantienes una conversación bastante sencilla y recuerda algunos aspectos que tú le vas contando[9].

La creación del *bot* se basa en estados. Dependiendo del estado en el que se encuentre el *bot* y del mensaje que le llegue activa una función u otra[10]. El esquema general del funcionamiento del *bot* se puede observar a continuación:

[9]Github con el código del bot de Telegram: https://github.com/python-telegram-bot/python-telegram-bot

[10]Notebook con todos los detalles sobre la creación de bots de Telegram: https://colab.research.google.com/drive/1BMGOYBeQN4es5atsGfdPfzLFt4y18CsQ?usp=sharing

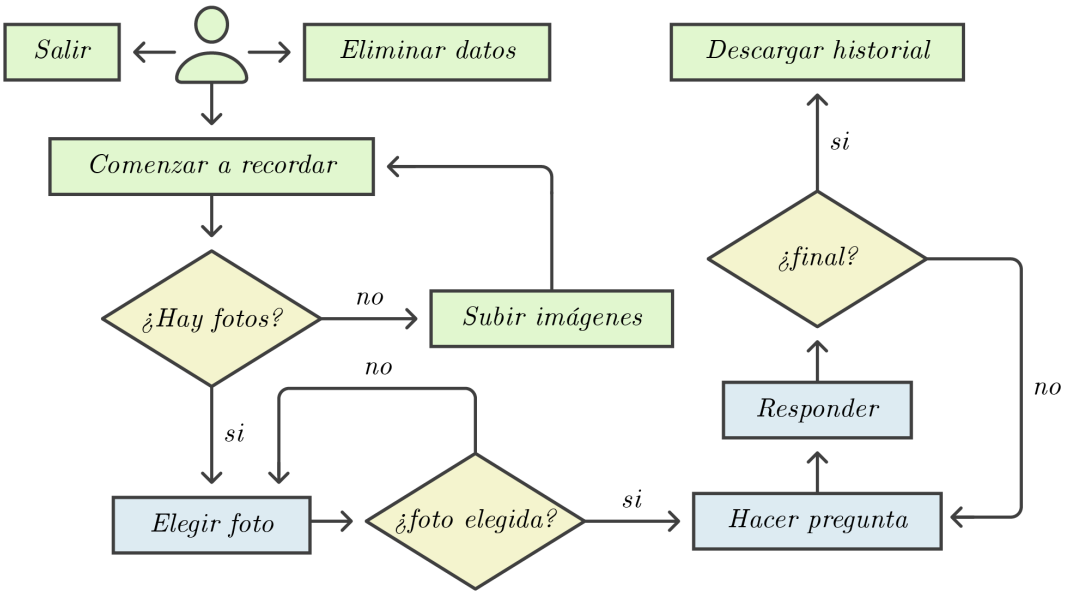

Figura 4.4: Esquema del funcionamiento del bot

Los 5 estados que dispone el *bot* son:

- **Comenzar terapia**: Este es el estado principal, en el que el usuario realizará la terapia. Consiste en ir mostrando al usuario imágenes previamente almacenadas para extraer preguntas y así ayudar al usuario a recordar.
- **Subir imágenes**: En este estado el usuario podrá subir las imágenes que quiera utilizar para la terapia. Podrá subirlas tanto de una en una como en grupo.
- **Descargar historial**: Una de las funcionalidades más interesantes es que almacenamos las preguntas y respuestas dadas por los usuarios. Así, en cualquier momento el usuario podrá descargarse un PDF que contiene las preguntas relacionadas con cada imagen y la respuesta que dio el usuario.
- **Eliminar datos**: En caso que el usuario quiera dejar de utilizar la aplicación podrá borrar todos sus datos (imágenes, respuestas, etc.).
- **Salir**: Aunque este botón no para el *bot* como tal, permite dar al usuario la opción de salir poniendo al *bot* en un estado de *stand-by*.

Para hacer la interacción con el *bot* más intuitiva transformamos el teclado convencional en botones. Es decir, que el usuario en lugar de indicar al *bot* las acciones que quiere tomar mediante comandos, solo tiene que pulsar el botón correspondiente,

lo que creemos que es mucho más fácil de utilizar para cualquier persona y sobre todo para las personas a las que está dirigida esta aplicación.

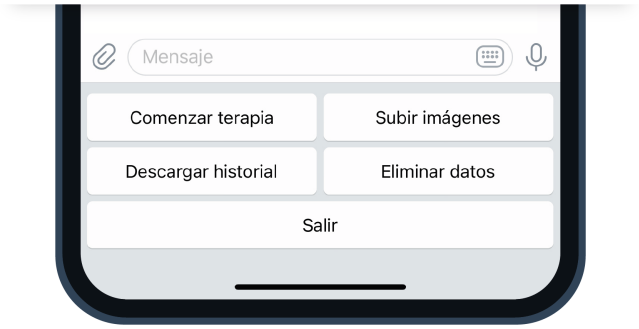

Figura 4.5: Interacción del usuario con el Bot

En cada estado del *bot* este teclado cambiará según las necesidades y volverá a convertirse en un teclado convencional cuando el usuario tenga que responder. Cada uno de los botones de los teclados son en realidad comandos de texto que mandamos en lugar de escribir. En cada estado creamos una función *callback* que se llamará en caso de recibir un mensaje específico. En caso de que el mensaje que llegue no sea ninguno de los preestablecidos se ignorará o, en caso de la terapia, se llamará a la correspondiente función *callback* para gestionar la respuesta del usuario.

4.4.1. Modulo de Traducción de idiomas

Una ampliación que pensamos en hacer desde el principio es permitir que el *bot* estuviera disponible en varios idiomas para así llegar a muchas más personas. Esto traía consigo diferentes problemas que tuvimos que solventar y pensar la mejor forma de hacerlo.

Al principio valoramos la idea de tener diferentes modelos en los principales idiomas (inglés, español y francés) pero tras no encontrar ningún *dataset* como los que buscábamos en ningún otro idioma que no fuese el inglés decidimos descartar esta idea. Tendríamos que entrenar el *bot* en inglés y generar las preguntas en este idioma.

Ahora teníamos el problema de traducir las preguntas al idioma por defecto del usuario. Para ello preferimos relegar este trabajo a una herramienta que lleva muchos años haciéndolo y cada vez con mejores resultados, Google Translate. Utilizando un módulo muy sencillo llamado `google-trans-new` nos permite traducir cualquier texto de un idioma a otro con una sola llamada a una función.

Ahora el problema era saber a qué idioma debemos traducir el texto. Por suerte la API de Telegram nos permite saber el idioma "preferido"del usuario en cada llamada al *bot* que se hace. Utilizamos esto para pasarle el código a la función y así obtener el texto traducido.

Para la interfaz y los mensajes decidimos ir un paso más allá para hacer que la experiencia con el *bot* sea aun mejor. En lugar de utilizar el traductor de Google implementamos distintos módulos que van a tratar con los mensajes y teclados especiales. De esta forma además nos permite tener todos los mensajes en un mismo sitio y separados por idiomas, lo que facilita muchísimo añadir, eliminar o modificar un mensaje.

En específico tenemos 3 archivos que utilizamos para esto: Translate, Messages y Keyboards.

4.4.1.1. Translate

Es el encargado de juntar todo. Tiene funciones que devuelven el teclado, el mensaje o la expresión regular necesaria en cada momento. Lo que hacemos es indicar el mensaje, teclado o expresión que queremos mediante una constante que hemos predefinido y el idioma en el que lo queremos devolver.

Por ejemplo, si queremos obtener el mensaje de bienvenida en español tendremos que pasarle la constante `WELCOME_MESSAGE` y en el idioma `es`.

Además, incorporamos una función para forzar el idioma del *bot* que consiste en ignorar el idioma del usuario y mostrar todo en el idioma elegido por defecto.

4.4.1.2. Messages y Keyboards

Contiene todos los mensajes que utiliza el *bot* en los distintos idiomas registrados. Existe una función por cada mensaje que devuelve el mensaje en todos los idiomas y luego una función general que, dado el idioma, elige finalmente el mensaje correcto a mostrar.

Con los teclados hace básicamente la misma operación solo que en este caso se trata de matrices de texto con la disposición de los diferentes teclados desarrollados.

4.4.2. Base de datos

Como ya hemos comentado anteriormente, uno de los objetivos era que una vez el usuario realizase la terapia pudiera consultar sus respuestas acompañado de

familiares o profesionales. La única forma de poder conseguir esto es mediante la persistencia de datos.

Decidimos que lo mejor era utilizar una BD relacional donde almacenaríamos datos sobre los usuarios, sus preguntas y respuestas. Las imágenes se almacenan en el servidor en una carpeta específica por cada usuario.

Al principio estuvimos debatiendo sobre que *Sistema Gestor de Bases de Datos* (SGBD) usar. Pensamos en utilizar uno convencional, pero tras evaluarlo decidimos utilizar la librería SQLite que nos proporciona lo que necesitamos de un SGBD convencional. SQLite es una librería de Python que nos permite tener una BD relacional en el propio servidor. Se trata de un pequeño archivo que puede ser de hasta 140 terabytes que funciona igual que una BD. Podemos realizar consultas y volcar datos de forma convencional utilizando las instrucciones *Structured Query Language* (SQL) comunes. Además, es mucho más rápido que leer de archivos almacenados y nos ahorra tener que depender de un SGBD. Para el objetivo y tamaño de este proyecto era más que suficiente.

Una vez decidida la tecnología que íbamos a utilizar diseñamos la BD como tal. El esquema general de la BD es el siguiente:

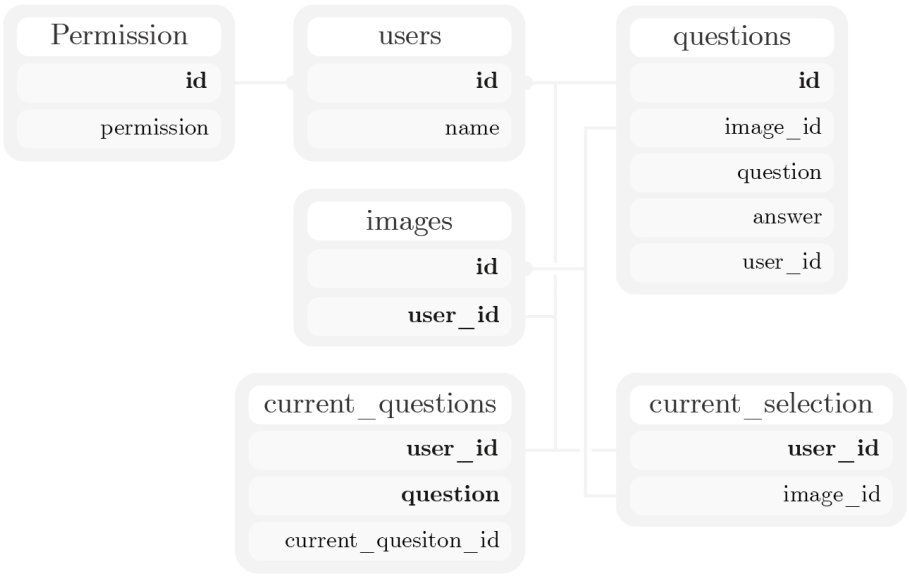

Figura 4.6: Esquema Entidad-Relación de la BD

Vamos a pasar a explicar cada uno de estos módulos y como los hemos utilizado:

4.4.2.1. Users

En este módulo como vemos en el esquema encontramos tres columnas el id, el nombre y el idioma. El primero hace referencia al identificador único de cada usuario. Guardaremos en la BD el usuario cada vez que inician sesión en nuestra aplicación creando la carpeta donde se almacenarán las imágenes.

4.4.2.2. Images

En este módulo guardamos el identificador del usuario y un identificador único incremental por cada foto del usuario. Esta tabla la utilizamos para poder saber qué imágenes dispone el usuario y si no ha subido ninguna foto todavía.

4.4.2.3. Questions

En esta tabla guardamos más información: el id del usuario, el id de la imagen en la que se hizo la pregunta, la pregunta como tal y la respuesta que dio el usuario. Utilizamos esta tabla para guardar las respuestas de los usuario para luego poder generar el pdf con su historial.

4.4.2.4. Current questions

A la hora de hacer la terapia nos encontramos con el problema de saber qué preguntas y qué imágenes habían elegido los usuarios para realizar la terapia en caso de que varios usuarios decidiesen utilizar la aplicación al mismo tiempo. Para solucionar este problema decidimos crear unas tablas que guardasen la información actual que estaba manejando cada usuario y que se eliminase al terminar la terapia. En el caso de las preguntas una vez generadas las guardamos en la BD con el id del usuario, la pregunta en cuestión y un identificador de pregunta para poder después sacarlas en orden. Esta tabla la utilizamos para poder ir mostrando las preguntas durante la terapia. Cada vez que el usuario responde a una pregunta, se muestra la siguiente y se elimina de la BD. Repitiendo esto hasta acabar con las preguntas.

4.4.2.5. Current selection

La solución para la imagen es similar a las preguntas. Guardamos el identificador del usuario y la imagen que ha seleccionado para saber en todo momento cuál es la imagen que está seleccionada para la terapia.

4.4.2.6. Permission

Cuando un usuario utiliza nuestra aplicación por primera vez se le preguntará si está de acuerdo con que se almacenen sus imágenes y sus preguntas. Esta respuesta la guardamos en la BD y se mantendrá aunque el usuario borre sus datos. De estas forma podremos conocer en todo momento si el usuario está de acuerdo con los términos de uso y saber si debemos volver a mostrarle la introducción o no. Guardamos el identificador de cada usuario y si está o no de acuerdo.

Capítulo 5

Ejemplos de uso

En esta sección vamos a comenzar por explicar como funciona la interfaz del *bot*, navegando por todas las opciones del *bot* y finalmente mostraremos algunos ejemplos de ejecución reales.

5.1. Remi

5.1.1. Primera interacción con el bot

La primera interacción que tengamos con el *bot* nos dará la bienvenida. Se trata de Remi, el robot que nos ayudará a tratar de recordar esos aspectos de nuestra vida que han quedado en el olvido pero que siguen presentes en nuestras fotografías. Este nos explicará brevemente de que trata y nos guiará en un pequeño tutorial para aprender a utilizar la interfaz y saber que hacer en todo momento.

Así pues la primera vez que iniciemos el *bot*, este nos explicará mediante mensajes de texto e imágenes cómo comunicarnos con él utilizando unos botones que aparecerán en la pantalla. Como en ocasiones estos botones pueden no aparecer (por como está implementada la propia interfaz de Telegram) el mismo tutorial muestra qué hacer para volver a ver los botones. De igual forma nos explica que en el momento que tengamos que responderle a sus preguntas estos botones se cambiarán por el teclado para que podamos escribir.

Por último, antes de poder comenzar a utilizar el *bot*, este nos informará de que tanto las imágenes que subamos como las respuestas que escribamos serán almacenadas. Para ello Remi nos pedirá consentimiento para poder almacenar estos datos.

Si nos negamos volverá a preguntar para cerciorarse de que sabemos que, sin este permiso, no lo podemos utilizar. Si volvemos a negarnos el Remi se despedirá y acabará la conversación. En el momento en el que concedamos el permiso el este pasará a mostrar el menú principal.

5.1.2. Bienvenida y menú principal

Una vez pasado el tutorial el *bot* no volverá a mostrarlo, en su lugar este nos dará la bienvenida refiriéndose a nosotros por nuestro nombre que obtiene al recibir la petición de `/start`. Al mismo tiempo también creamos la carpeta correspondiente para almacenar las imágenes utilizando el id del usuario que es único y nos lo proporciona la API de Telegram. A continuación Remi preguntará que queremos hacer. Es aquí donde el usuario podrá elegir una de las 5 posibles opciones mencionadas anteriormente.

Figura 5.1: Bienvenida del Bot

5.1.3. Subir imágenes

Para poder realizar la terapia el usuario necesita subir anteriormente las imágenes. Una vez el usuario seleccionado la opción de subir imágenes deberá subir las imágenes de forma digital pulsando el icono del clip y dando a subir imágenes. En caso de que las imágenes se encuentren en formato papel que es lo normal en imágenes antiguas se podrán subir sacando una foto directamente a la imagen en papel. Se intentará realizar la imagen de la mejor forma posible para evitar reflejos que puedan empeorar la predicción del bot. Para ello de la misma forma se pulsa sobre el botón del clip y se da a hacer foto.

Las imágenes se podrán subir de una en una o en un lote seleccionando varias desde la galería. Una vez el usuario haya terminado subiendo las imágenes podrá pulsar el botón terminar. El *bot* le dará las gracias como confirmación de que las imágenes se han subido de forma correcta.

Figura 5.2: Subida de imágenes

5.1.4. Comenzar terapia

Una vez el usuario ha subido sus imágenes podrá comenzar la terapia. En caso de que no haya subido ninguna imagen el *bot* se lo indicará.

Si todo ha ido bien Remi mostrará una imagen aleatoria entre todas las que

tiene subidas el usuario. Si no quiere hablar sobre esa foto el usuario podrá omitirla y Remi seleccionará otra foto de forma aleatoria hasta que el usuario esté conforme con la foto a utilizar.

Figura 5.3: Sin imágenes

Figura 5.4: Cambio de imagen

Una vez el usuario apruebe la foto comenzarán a generarse las preguntas. Lo primero que se hace es seleccionar una pregunta general de forma aleatoria. Estas preguntas son genéricas y están preseleccionadas por nosotros. Son preguntas de carácter general que pueden servir para cualquier fotografía. De esta forma el usuario empieza con una pregunta que le obliga a pensar sobre la imagen y mientras recuerda y responde se van generando las preguntas específicas de la imagen. Se generarán 10 preguntas en total de las que seleccionarán 4 de forma aleatoria. Hacemos esto para que en caso de que el usuario desee repetir una imagen no obtenga exactamente el mismo resultado.

Como el modelo lo tenemos en inglés las preguntas las generará en inglés. No obstante hemos utilizado una API de *Google Translate* que nos permite traducir de forma directa la pregunta generada por el modelo. Cuando el usuario manda una petición, en esta aparece el código de idioma del usuario lo que nos permite traducir las preguntas a cualquier idioma.

Una vez respondida a la pregunta Remi realizará otra, hasta un total de 5 en el que indicará al usuario que va a cambiar de imagen.

Figura 5.5: Preguntas de la terapia

Puede haber diferentes motivos por el que un usuario no quiera o pueda responder a una pregunta. Bien sea porque la traducción fue incorrecta, la pregunta generada no está bien construida, la persona no consigue recordar, etc. Para ello permitimos al usuario pasar de pregunta sin que esta quede registrada. Remi simplemente la ignorará y pasará a mostrar la siguiente pregunta en caso de que queden disponibles.

Además de poder pasar de pregunta también se permite terminar con la fotogra-

fía actual pulsando el botón cambiar fotografía o terminar con la sesión de terapia pulsando el botón terminar.

Figura 5.6: Saltando la pregunta

Figura 5.7: Saltando la imagen

5.1.5. Descargar historial

Uno de los puntos que nos parecía interesante es almacenar las respuestas que da el usuario para que este pueda luego descargarse sus respuestas y recordar qué había respondido. Además podría utilizarse junto con un terapeuta o un familiar que les permita ver qué respondió el paciente y analizar sus respuestas.

Hemos implementado un botón para que el usuario pueda descargarse sus preguntas y sus respuestas para cada imagen. En lugar de mostrarlas por Telegram preferimos generar un pdf ya que de esta forma se puede imprimir y es más cómodo que revisar una conversación de chat.

El pdf mostrará cada imagen que se ha utilizado en la terapia junto con la pregunta formulada y la respuesta dada por el usuario. Se enviará a la conversación de Telegram como documento y el usuario podrá enviárselo por email o imprimirlo según lo prefiera.

Capítulo 6

Modificaciones, Resultados y Limitaciones

En este capítulo explicaremos los resultados obtenidos en los diferentes experimentos realizados, así como las principales limitaciones de nuestra red neuronal y nuestro trabajo en general.

6.1. Modificaciones

Con el objetivo de obtener los mejores resultados posibles realizamos una serie de modificaciones y experimentos en nuestro proyecto para poder comparar los resultados y elegir la mejor opción.

A continuación expondremos los principales experimentos realizados.

6.1.1. GloVe Embbedings

GloVe es un modelo preentrenado para la representación de palabras distribuidas que obtiene las representaciones vectoriales de estas. Las palabras se asignan a un espacio vectorial donde las similares están agrupadas y aquellas que son diferentes se separan.

La ventaja de utilizar Glove es que no solo se basa en el contexto local de las palabras, sino que incorpora la co-ocurrencia global en una matriz, por lo que podemos derivar relaciones semánticas entre palabras a partir de dicha matriz.

En nuestro modelo, asignaremos todas las palabras de nuestra pregunta a un vector de trescientas dimensiones usando GloVe y después crearemos una matriz de dimensiones (tamaño_vocabulario, 300) formada por nuestro vocabulario y el vector de Glove.

Finalmente, a la hora de entrenar el modelo, tenemos que tener en cuenta que no queremos volver a entrenar los pesos en nuestra capa de embedding, ya que contamos con los vectores GloVe previamente entrenados. Para ello simplemente cambiamos la propiedad "trainable"de nuestra capa embedding a false y le metemos los pesos correspondientes.

```
model.layers[2].set_weights([embedding_matrix])
model.layers[2].trainable = False
```

6.1.2. Aumento del conjunto de datos de entrenamiento

Con los *datasets* que encontramos al inicio del proyecto decidimos realizar pequeños experimentos que tienen que ver con la cantidad de datos que utilizamos para entrenar nuestro modelo.

Decidimos entrenar nuestro modelo varias veces con diferentes conjuntos de datos. Primero entrenamos el modelo únicamente con tres de los *datasets* los cuales fueron Coco, Bing y Flickr. Después decidimos añadir el *dataset* mas grande de todos (Microsoft Coco) al entrenamiento.

6.1.3. Aumento del numero de *Epochs* de entrenamiento

Al igual que con la cantidad de datos de entrenamiento, también hicimos experimentos similares con el número de veces que estos *datasets* iban a pasar a través de nuestra red neuronal durante el proceso entrenamiento.

Hicimos entrenamientos de 5, 10 y 15 *epochs* con los datos de Coco, Bing y Flickr y un entrenamiento de 7 *epochs* para el conjunto de datos global (añadiendo Microsoft Coco a los anteriores).

El entrenamiento de nuestro modelo con todos los datos era un proceso que requería mucho tiempo y recursos, por lo que solo pudimos realizar un entrenamiento de 7 epochs. Además todos estos experimentos nombrados en las anteriores secciones fueron realizados con las dos variantes del modelo, el modelo con GloVe y sin Glove.

6.2. Resultados

En esta sección vamos a comparar la precisión y calidad de las preguntas obtenidas tras entrenar nuestra red neuronal después de realizar los experimentos explicados anteriormente. Los ejemplos explicados a continuación corresponden a estos experimentos:

- **Ejemplo 1 y 2**: Entrenamiento del la red con los datasets de Coco, Flickr y Bing realizando quince *epochs*. En estos ejemplos se usarán dos fotos diferentes.
- **Ejemplo 3**: Entrenamiento del la red con los datasets de Coco, Flickr y Bing realizando quince *epochs* pero utilizando GloVe Embeddings.
- **Ejemplo 4**: Entrenamiento del la red con los *datasets* de Coco, Flickr y Bing pero realizando únicamente cinco *epochs* de entrenamiento.

6.2.1. Ejemplo 1

Cuando iniciamos el *bot* Remi nos da la bienvenida y nos pregunta qué queremos hacer. En esta ejecución hemos configurado el *bot* para que hable en inglés para poder apreciar la calidad de las preguntas sin pasar por el módulo de traducción. Así podemos valorar como esta actuando el modelo. Empezamos subiendo una foto de un concierto.

Una vez la he subido pulso sobre el botón de empezar a recordar, confirmo que quiero hablar sobre esta foto y Remi empieza a formular las preguntas:

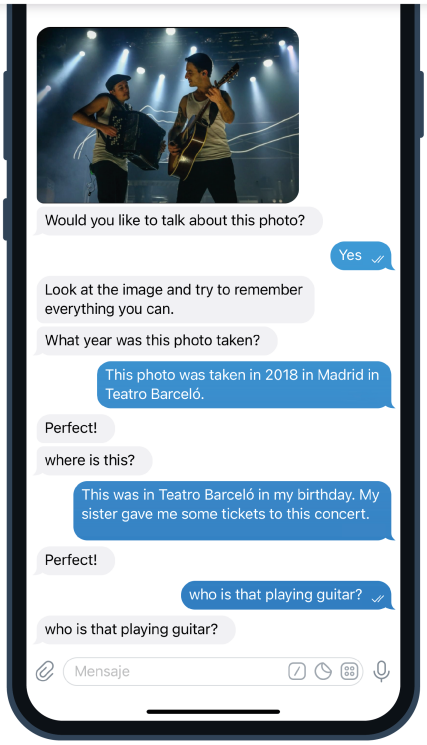

Figura 6.1: Ejecución ejemplo 1 - 1

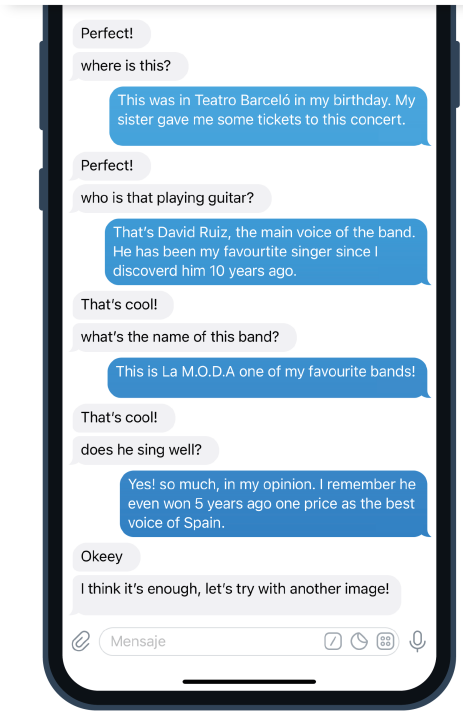

Figura 6.2: Ejecución ejemplo 1 - 2

Las dos primeras preguntas son acerca del momento y el lugar en el que se tomó la foto. Hasta ahora parecen preguntas genéricas que valdrían para prácticamente cualquier imagen. En las siguientes preguntas parece que ya ha identificado los instrumentos y enfoca las preguntas dentro del contexto musical.

En general en este ejemplo las preguntas generadas son bastante buenas. A veces al traducir las preguntas se pierde la calidad, por ejemplo con esta foto se tradujo *"who is that playing the guitar?"* como "¿quién es él jugando la guitarra?". Aunque en otras muchas ocasiones el traductor hace su función correctamente si que añade un extra a la probabilidad de fallo del modelo.

6.2.2. Ejemplo 2

Vamos a realizar la misma prueba con una imagen de elefantes.

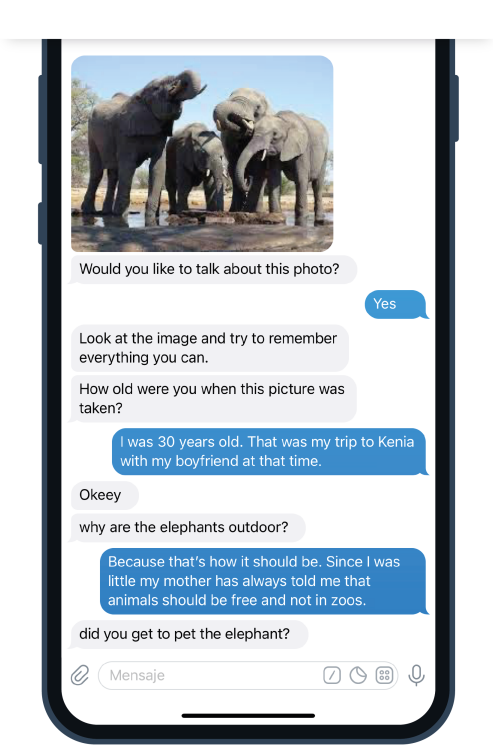

Figura 6.3: Ejecución ejemplo 2 - 1

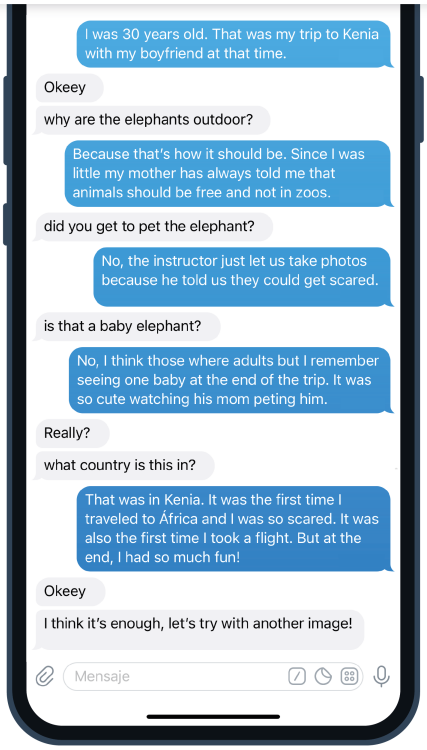

Figura 6.4: Ejecución ejemplo 2 - 2

Por como hemos implementado el *bot* sabemos que la primera pregunta será de carácter temporal. Después de está el modelo identifica tanto a los elefantes como el hecho de que no parecen estar en cautividad.

Tras eso nos hace un par de preguntas más sobre los elefantes, si los domesticamos o si alguno de ellos es una cría y finalmente pregunta por el país donde ocurrió esto.

Las preguntas generadas sobre esta imagen también son bastante buenas. La pregunta sobre si hemos conseguido domesticar al animal ("*Did you get to pet the elephant?*") no es muy natural pero si guarda relación lo que vemos en la foto y podría ser útil para probar el estado del paciente.

6.2.3. Ejemplo 3

En este ejemplo vamos a utilizar la misma foto de los elefantes para ver un poco la comparación de las preguntas. Como hemos dicho anteriormente se trata de 15 épocas con Glove.

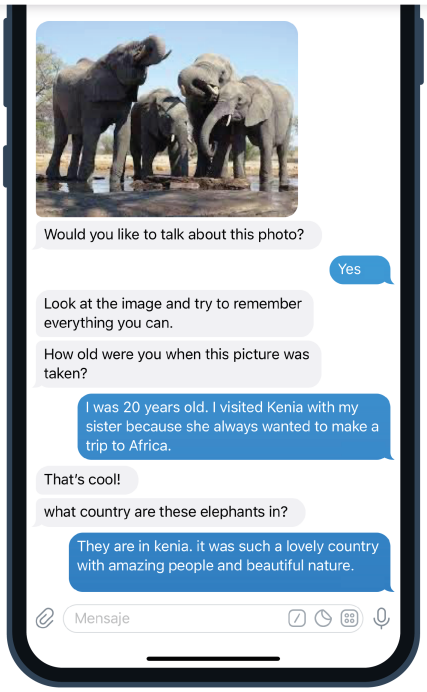

Figura 6.5: Ejecución ejemplo 3 - 1

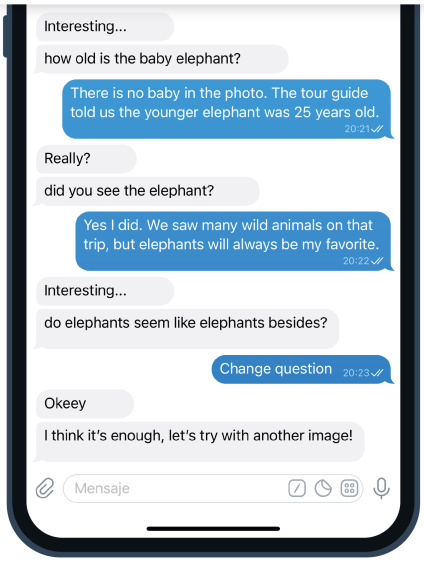

Figura 6.6: Ejecución ejemplo 3 - 2

Con este entrenamiento conseguimos que el modelo diese buenos resultados sin embargo en varias ocasiones las preguntas carecen de sentido. En este ejemplo aparece la pregunta *"do elephants seem like elephants besides?"*. Parece que por intentar realizar preguntas más complejas acaba formulándolas mal.

Quizá más épocas de entrenamiento hubiesen solventado este problema pero cuando llegamos a 15 y vimos que no se solucionaba optamos por dejar de utilizar Glove.

6.2.4. Ejemplo 4

Para este ejemplo vamos a volver a usar la misma foto para poder comparar los distintos entrenamientos. Esta vez es un entrenamiento de 5 épocas que nos dio muy buenos resultados.

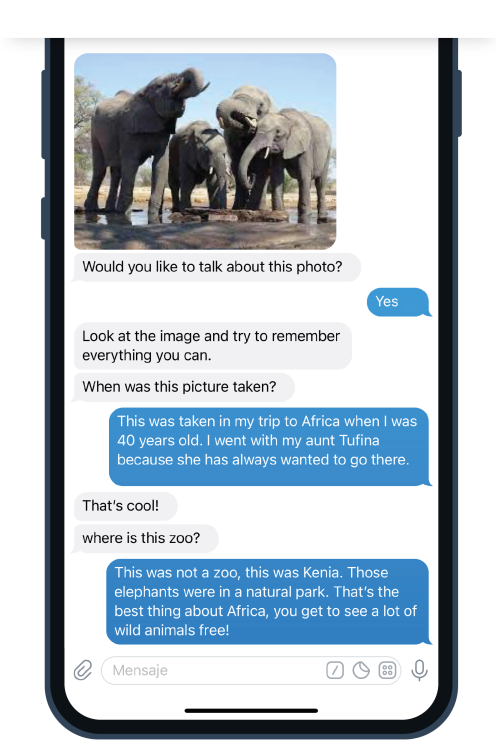

Figura 6.7: Ejecución ejemplo 4 - 1

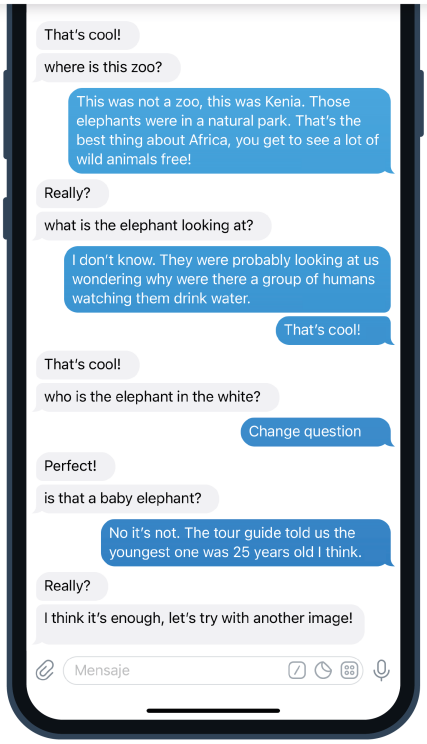

Figura 6.8: Ejecución ejemplo 4 - 2

En este ejemplo se identifica bien a los elefantes y casi todas las preguntas tienen sentido y están bien formuladas. La excepción es la pregunta *"who is the elephant in the white?"* que parece que ha confundido la palabra *white* por otra.

6.2.5. Resumen de los resultados

Como hemos podido observar en los resultados mostrados anteriormente, todos los experimentos realizan preguntas diferentes en función de las tecnologías utilizadas y el número de épocas.

Cabe destacar que, al usar el *bot* en Castellano, la calidad de las preguntas disminuye ligeramente debido a que el módulo de traducción no es perfecto (como explicamos en la sección de limitaciones) por lo que decidimos realizar los experimentos y la comparación de los resultados en inglés para poder apreciar la calidad real de las cuestiones generadas por la red.

En los ejemplos 1 y 2, las preguntas son de mejor o peor calidad dependiendo de la imagen de entrada que aporta el usuario. Esto se debe a que en el conjunto de

datos de entrenamiento posiblemente existan mas imágenes similares a la del ejemplo 1 que a la del ejemplo 2, por lo que la red ha podido aprender a reconocer mejor los elementos de este tipo de imágenes.

En cuanto al ejemplo 3, podemos observar que la calidad de las preguntas ha disminuido al usar GloVe, ya que su construcción sintáctica es muy rara y a veces no tienen sentido. Podemos ver que reconoce los elementos de la imagen, pero no construye la pregunta como se debería.

Las preguntas del ejemplo 4 también son de peor calidad con respecto a las de los ejemplos 1 y 2. Esto es debido principalmente a que la red neuronal solo fue entrenada con los mismos datasets durante cinco epochs, es decir, 10 epochs menos respecto a los primeros ejemplos.

Para que nuestro bot funcione es necesario elegir uno de estos modelos obtenidos después de realizar los experimentos. Al comparar los resultados de cada uno de los experimentos **nos decantamos por usar el modelo entrenado durante quince epochs con los datasets de Coco, Flickr y Bing** (correspondiente a los ejemplos 1 y 2) ya que como se puede observar en los ejemplos mostrados arriba es el modelo con mejores resultados.

6.3. Limitaciones

Como hemos podido comprobar en el apartado de resultados, las preguntas generadas por nuestros modelos no son siempre las mejores. Esto se debe a que nuestro modelo presenta una serie de limitaciones importantes que explicamos a continuación.

- **Calidad de los datos recolectados**: La calidad de los datos de entrenamiento es uno de los aspectos mas importantes. Cuanto mayor sea la calidad de dichos datos, mayor será la calidad de los resultados obtenidos.

  Nos encontramos con muchas limitaciones a la hora de encontrar datos que nos sirviesen para nuestro proyecto ya que estos tenían que estar compuestos por imágenes y preguntas asociadas a estas además de tener que presentar un contexto de terapia de reminiscencia o terapias similares reales.

  Además, los datos que finalmente obtuvimos no fueron de la mejor calidad. Tuvimos que realizar un pequeño procesado de las preguntas y no todas las imágenes mostraban cosas fácilmente reconocibles o su resolución no era la mejor.

- **Tiempo y recursos**: El entrenamiento de la red neuronal con un gran volumen de datos requiere el uso de tecnologías avanzadas de procesamiento gráfico para

optimizar dicho entrenamiento. Nosotros nos ayudamos de un servidor donde pudimos realizar varios entrenamientos.

Aun así, el tiempo empleado para cada uno de estos era bastante elevado, por lo que esto nos limitó a la hora de realizar más experimentos para poder intentar mejorar nuestros resultados.

- **Traducciones erróneas**: Esta limitación tiene que ver también con la calidad de los datos. Fue imposible encontrar conjuntos de datos significativos que estuviesen en nuestro idioma, el Castellano.

  Esto hizo que tuviésemos que recurrir a datos y preguntas formuladas en inglés lo que provoca que la red de sus resultados también en este idioma y tengamos que utilizar un módulo de traducción de Google para obtenerlas en español. Esto añade pequeños fallos, ya que las traducciones del inglés al español realizadas por dicho módulo no son perfectas.

Todas estas limitaciones provocan que las preguntas obtenidas en ocasiones no sean las mejores, ya que puede que no estén perfectamente formuladas o no sean preguntas útiles para estimular la memoria del paciente.

Capítulo 7

Conclusiones y trabajo futuro

Comenzamos el proyecto con el estudio de las técnicas de Deep Learning. Pudimos probar y testear soluciones parecidas a las que nosotros queríamos implementar y previsualizar nuestro trabajo. En específico pudimos ver como funcionaban soluciones como un clasificador de imágenes que conseguía clasificar en la mayoría de las veces de forma correcta un objeto. Este sería nuestro punto de partida ya que en definitiva nosotros queríamos obtener la información de la imagen para luego poder extraer preguntas de ella.

Después de crear nuestra red neuronal realizamos varios estudios con diferentes conjuntos de datos. Obtuvimos modelos diferentes con los que probamos utilizando el mismo conjunto de imágenes y fuimos generando nuestras propias conclusiones sobre que modelo era mejor en base a los resultados obtenidos.

Este trabajo tiene muchas líneas de mejora y ampliación. Por un lado podríamos utilizar un modelo de atención para generar mejores preguntas y mejor construidas. Los mecanismos basados en la atención se están volviendo cada vez más populares en el aprendizaje profundo, porque pueden enfocarse dinámicamente en las diversas partes de la imagen de entrada mientras se producen las secuencias de salida.

También podríamos buscar la forma de obtener datos de terapias reales y generar nuestros propios *datasets* para entrenar el modelo, incluso con datos en español para que las preguntas sean mejores y de mayor calidad. Además se podría ampliar para utilizar más elementos multimedia como vídeos y audios que permitan extraer otro tipo de preguntas. En caso de poder acceder a recursos más potentes de procesamiento gráfico, otra mejora posible sería entrenar la red neuronal durante más *epochs* y con conjuntos de datos más amplios. Esto junto con la mejora de la calidad de los datos mencionada anteriormente mejoraría en gran medida la calidad de las

predicciones realizadas por el modelo.

Se podría añadir una nueva capa de interactividad haciendo que el *bot* responda de forma más específica a la respuesta que da el usuario, dándole *feedback* en lugar de mostrar una respuesta genérica como se hace actualmente. Para ello habría que crear otra red neuronal que procesase la respuesta del usuario y crease una frase en función de esta.

También podríamos mejorar mas aún la interfaz interactiva del *bot*, adaptándola a todas la necesidades que puedan llegar a tener las personas que vayan a utilizarlo. Para este fin sería de utilidad realizar pruebas de uso con personas de la tercera edad para recopilar requisitos.

La mayoría de las personas con problemas de memoria suelen ser personas de avanzada edad, por lo que tener botones más grandes, explicaciones más detalladas u otro tipo de elementos podrían ser de gran ayuda para estas personas.

Como conclusión podemos afirmar que, teniendo en cuenta todas las limitaciones y posibles mejoras que tiene nuestro proyecto, los resultados obtenidos son buenos tal y como hemos mostrado y explicado en el capítulo anterior. Con estos resultados la persona con problemas de memoria puede iniciar una pequeña terapia de reminiscencia individualmente, para cuando el profesional que se encarga de ella no esté disponible. Además puede guardar y descargar los resultados de la terapia para que el profesional tenga constancia de ella.

Chapter 7

Conclusions and future work

We started the project with the study of Deep Learning techniques. We were able to test solutions similar to the ones we wanted to implement and preview our work. Specifically, we could see how solutions such as an image classifier worked and managed to classify an object correctly. This would be our starting point because we wanted to obtain the information from the image in order to then be able to extract questions from it.

After creating our neural network we conducted several studies with different data sets. We obtained different models, tested them using the same set of images and then generated our own conclusions about which model was better based on the results obtained.

This work has many ways of improvement and expansion. On the one hand, we could use an Attention model to generate better constructed questions. Attention-based mechanisms are becoming increasingly popular in deep learning, because they can dynamically focus on the different parts of the input image while producing the output sequences.

We could also find a way to obtain data from real therapies and generate our own *datasets* to train the model, even with data in Spanish. This will make the questions better and of higher quality. It could also be expanded to use more multimedia elements such as videos and audios that allow extracting other types of questions. In case of being able to access to more powerful graphic processing resources, another possible improvement would be to train the neural network for more *epochs* and with larger data sets. This together with the aforementioned data quality improvement would greatly improve the quality of the predictions made by the model.

A new layer of interactivity could be added by making the *bot* respond more specifically to the response that the user gives, giving *feedback* instead of displaying a generic response as it is currently done. To do this, another neural network would have to be created to process the user's response and create a sentence based on it.

We could also further improve the interactive interface of the *bot*, adapting it to all the needs that people who are going to use it may have. For this purpose, it would be useful to carry out use tests with the elderly to gather requirements.

Most people with memory problems tend to be elderly, so having bigger buttons, more detailed explanations or other types of elements could be a great help for these people.

As a conclusion we can affirm that, taking into account all the limitations and possible improvements that our project has, the results obtained are good as we have shown and explained in the previous chapter. With these results, the person with memory problems can start a small reminiscence therapy individually, using the bot when the professional who takes care of it is not available. You can also save and download the results of the therapy so that the professional can have a record of it.

Capítulo 8

Aportaciones individuales al proyecto

8.1. Alejandro Aizel Boto

Desde que comenzó el proyecto he participado en todas las tareas del mismo tanto de programación, como de organización, ideas y documentación. Al principio del proyecto realicé al igual que mis compañeros, los cursos de Deep Learning necesarios para poder llevar a cabo el proyecto. Al mismo tiempo realizaba otro curso sobre Latex para poder escribir la memoria y sobre *bots* de Telegram para poder implementar nuestra solución en Telegram.

Antes de comenzar a hacer nuestra implementación participé en la búsqueda y documentación de soluciones similares y de recursos intentando entender qué se hacía y cómo se hacía. Una vez que mis compañeros y yo teníamos claro qué queríamos hacer comenzamos con la implementación.

Comencé realizando la implementación del *bot* de Telegram que se encarga de recibir las llamadas desde la aplicación para poder realizar lo que se solicita. Implementé los diferentes estados y las funciones *callback* que más adelante se utilizarían para llamar a las demás funciones. Además, diseñé el personaje de la aplicación y las distintas expresiones que adquiere. Una vez terminado pasé todo el código a un *notebook* de Python donde explicaba el funcionamiento del *bot* en detalle para tenerlo documentado y poder consultarlo en cualquier comento.

Una vez que el *bot* ya estaba terminado, aunque sin ninguna funcionalidad todavía, realicé la búsqueda de *datasets*. Encontré distintos *datasets* con formatos diferentes por lo que realicé un módulo para poder tratar estos datos. El objetivo era obtener un solo conjunto de datos formateados de la misma forma para si poder

luego entrenar sin preocuparnos de nada. Este módulo al igual que el anterior, lo pasé a un *notebook* donde lo documenté para poder revisarlo posteriormente.

Una vez teníamos esto preparado, junto con mis compañeros realizamos la implementación como tal del proyecto. Generamos la red neuronal y una vez en funcionamiento lo probamos y arreglamos errores en conjunto.

Una vez finalizado esto desarrollé el módulo translate. Este módulo consta de 3 ficheros con los mensajes y teclados que se mostrarán en el idioma del usuario. Sutituí todas las frases que habíamos puesto en el bot por llamadas a funciones que devolvían el mensaje correcto en el idioma correcto. Al igual que los módulos anteriores describí su funcionamiento en 3 notebooks (uno por cada fichero) donde explico como utilizar el módulo.

Una de las ideas que queríamos implementar era la persistencia de datos. Para ello decidimos utilizar SQLite para poder almacenar los datos. Diseñé y monté la base de datos utilizando esta tecnología y escribí su funcionamiento básico en un notebook para conocer por parte de todos su funcionamiento. También organicé las carpetas y cómo se iba a guardar la información y las imágenes. Realicé también algunos daos encargados de conseguir lo que queríamos.

Por último con respecto al código desarrollé el módulo *get history* que se encarga de generar un pdf con las preguntas y respuestas de los usuarios y enviarlo por Telegram al usuario que lo ha solicitado. Además realicé también los daos que se necesitaban para poder realizar esto.

Con respecto a la memoria participé de forma directa o indirecta en todos los apartados escribiendo o completando. Además reestructuré el proyecto y lo preparé para que sea más cómodo realizar la memoria. Específicamente he participado en los siguientes módulos:

- **Introducción**: En la introducción corregí errores, reescribí algunos párrafos y añadí bastante información.
- **Deep Learning y Trabajo Previo**: Completé ligeramente la parte de trabajo previo y realicé gran parte de este punto. En específico hasta el punto 2.4.4 junto con todos los gráficos.
- **Herramientas, Tecnologías y Metodología de Trabajo**: En este capítulo realicé pequeñas correcciones y pequeñas aportaciones en el punto 3.1.1.
- **Remi - Bot para terapia dereminiscencia basado en VQG**: En este punto realicé aportaciones en la introducción y el punto 4.2 y realicé el punto 4.2.1, donde se explica el módulo de datasets y el 4.4 donde se explica el bot como tal.

8.2. Roberto Portillo Torres

Desde que empezamos a trabajar en este proyecto hemos realizado la mayoría de tareas del mismo conjuntamente. Aunque ahora vaya a citar las aportaciones que he realizado de manera más independiente en muchas de ellas me han ayudado mis compañeros siempre que lo he requerido.

En cuanto concertamos la primera reunión con Alberto nos recomendó que empezásemos a formarnos en *Deep learning* ya que la formación que teníamos hasta el momento era bastante básica. A pesar de haber cursado varias asignaturas sobre aprendizaje automático en la carrera es cierto que necesitaba ampliar mi conocimiento para afrontar el trabajo. Por ello nos pusimos durante unos meses a formarnos con los cursos de Andrew NG que ya hemos mencionado unas cuantas veces en esta memoria.

Además Alejandro propuso utilizar github para poder gestionar las versiones del código. Como yo no había trabajado aún con ello dedique una semana a informarme y completar algunos tutoriales sobre modos de uso y estándares. Pocas semanas después realicé la misma formación con Latex cuando Alberto nos recomendó encarecidamente que lo utilizásemos para la memoria.

A medida que iba acabando los cursillos y formándome una visión de como sería implementar un modelo basado en aprendizaje profundo me puse a investigar sobre cuál sería el entorno más cómodo para nosotros. Para esto realicé algunos tutoriales de Pytorch y leí algunos artículos sobre modelos de descripción de imágenes en Keras.

Una vez tomadas las decisiones sobre las tecnologías que íbamos a utilizar empezamos a construir nuestro modelo. La mayoría del desarrollo se hizo en sesiones conjuntas durante los fines de semana y a lo largo de la semana cada uno solucionaba errores menores por su cuenta.

Cuando tuvimos un código funcional y ya sabíamos la forma que debía tener el entorno en el que se fuese a ejecutar el *bot* me propuse instalar todo lo necesario en el servidor *holstein* que nos había cedido la universidad. Esta tarea a priori parecía más sencilla de lo que resultó. Hubo varios conflictos con librerías previamente instaladas que tuvimos que reinstalar para asegurarnos de que los distintos paquetes estaban en la versión que requería nuestro código.

Como de los integrantes del proyecto yo era el que disponía de mayor potencia computacional así como de la mejor conexión a internet se me asignó la tarea de descargar los datasets y prepararlos para el entrenamiento. Esto realmente no supuso un gran esfuerzo aunque si que tuvo mi máquina personal ocupada por unos días. Una vez los descargué, los convertí al formato correcto usando el script de Python que desarrolló Alejandro para este fin. Después los deje colgados tanto en nuestro

Google Drive como en el servidor *holstein*.

El *holstein* tenía todo listo para empezar a entrenar pero cuando íbamos a ello apareció un error nuevo que no aparecía en ninguna de la máquinas locales de los integrantes del grupo. Investigué y descubrí que el problema era que la CPU del servidor no soportaba Tensorflow. Me puse en contacto con Alberto y nos dijo que se lo comentaría al gestor de los servidores para que nos proporcionase acceso a una máquina más potente.

Mientras esperábamos el acceso a un servidor nuevo investigué una forma de ejecutar comandos Bash en los cuadernos de *Google Collaboratory* para poder ejecutar *scripts* almacenados en nuestro *Google Drive* compartido. Esto nos sirvió para entrenar durante un tiempo aunque cuando conseguimos acceso al nuevo servidor descartamos la idea de ejecutar en *Google Collaboratory* ya que era bastante más lento.

Más adelante nos dimos cuenta de que el formato del dataset Bing era bastante distinto a los demás y que muchos de sus datos no se estaban usando en el entrenamiento. En unas pocas horas de revisión del código junto con mi compañero Daniel conseguimos dar con el problema.

Cuando tuvimos un modelo lo suficientemente entrenado como para generar preguntas decentes y la interfaz de Telegram estaba operativa, me puse a buscar la forma de dejar la instancia del *bot* ejecutándose en el servidor sin necesidad de que tuviésemos que estar conectados. Encontré la aplicación *screen* y con ella dejé algunas sesiones en ejecución constante, una ejecutando el *bot* y otras para entrenar el modelo.

De cara a la memoria he recabado información y redactado los siguientes puntos:

- **Deep learning y trabajo previo:** En este apartado he explicado las partes relacionadas con la arquitectura *encoder-decoder*, *Word embeddings*, el modelo VQG e idea base.
- **Experimentos, resultados y limitaciones:** En este apartado expongo ejemplos de uso que reflejen lo mejor posible el funcionamiento del *bot*.
- **Bibliografía y referencias:** He añadido referencias por toda la memoria y añadido las entradas de la bibliografía.

También he revisado la estructura y la ortografía así como completado información por todos los puntos de la memoria.

8.3. Daniel Sanz Mayo

Durante todo el curso he participado en la mayoría de tareas necesarias para el desarrollo del proyecto. Mis conocimientos sobre Machine y Deep Learning eran básicos al inicio de curso, por lo que los primeros meses invertí gran parte de mi tiempo en documentarme y aprender nuevos conceptos para poder entenderlos mejor y aplicarlos en el trabajo.

A la hora de buscar datos con imágenes y preguntas relacionadas realicé una búsqueda encontrando nuevas páginas que contenían numerosos datos de gran ayuda para el entrenamiento de nuestro proyecto. He intervenido de forma activa en la programación de los modelos de la red neuronal (Encoder, Decoder, Beam Search ...), es decir, la parte clave de nuestro trabajo. También colaboré a la hora de realizar correcciones y mejoras sobre los modelos, como por ejemplo la inclusión de Glove Embeddings como alternativa y posible mejora para los resultados de nuestra red. Además, realicé gran parte del entrenamiento de los modelos creados para después poder probar el correcto funcionamiento de estos tanto en el servidor en el cual nos apoyamos para hacer entrenamientos grandes y simultáneos como en Google Colab.

En cuanto al *Bot* de Telegram ayudé en el modulo de mensajes añadiendo nuevos mensajes, teclados y refactorizando código general del modulo del *Bot*. En cuanto a la base de datos SQLite que presenta nuestro proyecto, cooperé en la creación de los diversos módulos que la componen programando los diferentes *daos* correspondientes y generando las consultas necesarias para la extracción y almacenamiento de información en la base de datos.

Respecto a la Memoria he escrito y aportado en diversos módulos:

- **Resumen**: Escribiendo el apartado del resumen tanto en español como inglés.
- **Introducción**: Escribiendo el apartado de la introducción tanto en español como inglés, donde introduzco los problemas actuales respecto a las enfermedades neurodegenerativas y el enfoque que vamos a dar a nuestro proyecto con el objetivo de ayudar a las personas con Alzheimer o problemas de memoria simulando una terapia de reminiscencia.
- **Deep Learning y Trabajo Previo**: Describiendo brevemente los cursos realizados por los miembros del grupo para aumentar nuestros conocimientos sobre Deep Learning.
- **Herramientas, Tecnologías y Metodología de Trabajo**: Este capítulo fue enteramente redactado por mi, describiendo todo lo relacionado con las aplicaciones y herramientas que hemos utilizado durante nuestro trabajo así como

nuestra forma de trabajar y organización durante el curso. Me encargo de enumerar y explicar en este apartado de la memoria estas aplicaciones esenciales a la hora de realizar nuestro trabajo y creo un pequeño diagrama de Gantt para resumir las principales tareas del proyecto y su duración durante el curso.

- **Modelos**: Encargándome de escribir los apartados 4.1 y 4.3 correspondientes a la obtención de los conjuntos de datos y el modelo VQG (Encoder, Decoder y Beam search) respectivamente. En estos módulos aporto una descripción bastante detallada de los modelos pertenecientes a nuestra red neuronal con pequeñas lineas de código y figuras descriptivas.
- **Experimentos, Resultados y Limitaciones**: En este capítulo escribí la parte de experimentos y de limitaciones, describiendo las diferentes modificaciones que habíamos hecho en el proyecto para obtener mejores resultados y las limitaciones que nos habíamos encontrado durante este.
- **Conclusiones y Trabajo futuro**: Escribiendo la conclusión y pequeños párrafos sobre trabajo futuro.

Apéndice A

Manuales de uso

A.1. Manual de uso del módulo Dataset

Este módulo es el que se encarga de descargar y formatear los *datasets* de forma correcta. Incluye también funciones que se utilizarán posteriormente para tratar los *datasets* que tengamos. Al ejecutar este módulo directamente nos saldrá un menú donde podremos elegir qué queremos hacer:

```
Choose the action to take:
   1. Download datasets.
   2. Load datasets.
   3. Format mscoco database.

Choice:
```

La primera opción nos permitirá descargar las imágenes y las preguntas desde varios archivos *Comma Separated Values* (CSV) que se incluyen. La segunda opción es simplemente de prueba ya que no tiene sentido cargar los datos si no los vamos a utilizar. Vamos a explicar la primera opción:

A.1.1. Descargar los datasets

Si pulsamos la primera opción `Choice: 1` nos va a volver a salir un menú que nos preguntará qué queremos hacer:

```
Choose the action to take:
   1. Download all datasets.
   2. Download a specific dataset.

Choice:
```

La primera opción nos permite descargar todos los *datasets* que hay disponibles en el directorio `dataset/data_to_download`. La segunda opción nos va a permitir elegir qué *dataset* y que conjunto de datos queremos descargar.

En caso de elegir la primera opción comenzará *dataset* a *dataset* a descargar (no recomendamos esta opción ya que tarda mucho en descargar). Si pulsamos la segunda opción nos preguntará qué *dataset* queremos descargar y qué conjunto. Vamos a decirle que queremos el *dataset* coco y el conjunto de datos *validation*:

```
Enter the name of the dataset to download (coco, flickr, bing): coco
Enter the name of the set to download (train, validation, test):
    validation
```

Si todo va bien el *dataset* comenzará a descargarse:

```
[Downloading images from the dataset coco_validation]
[DONE] Image 1/1249 with id coco_519601 correctly downloaded.
[DONE] Image 2/1249 with id coco_394518 correctly downloaded.
[DONE] Image 3/1249 with id coco_161503 correctly downloaded.
[DONE] Image 4/1249 with id coco_518601 correctly downloaded.
...
```

Puede ocurrir que haya fotos que no se descarguen debido a que ya no están disponibles. En caso de que esto ocurra se mostrará un mensaje de error en esa imagen. Esto no es un problema ya que simplemente no añade esta fotografía al *datset*:

```
[Downloading images from the dataset bing_test]
[ERROR] Image 1/1227 with id
    bing_214ec47b-07fa-4466-9a74-792d0fa08252 is not available to
    download.
[DONE] Image 2/1227 with id bing_bee043fb-4c71-4921-bb78-9d7c0127ac53
    correctly downloaded.
[DONE] Image 3/1227 with id bing_737848b3-926c-46d6-81e9-a86172986868
    correctly downloaded.
[ERROR] Image 4/1227 with id
    bing_10d52ed2-a7e5-4484-8a31-6a516e7038bb is not available to
    download.
[DONE] Image 5/1227 with id bing_4ab8e43f-a6a8-415f-94f8-e054397a249f
    correctly downloaded.
...
```

En caso de desconexión no hay problema, ya que cuando se vuelva a poner a descargar un *dataset* este comenzará a descargar donde se quedó. Eso si, las imágenes que anteriormente dieron error las colverá a intentar descargar. El archivo con las preguntas no se guardará hasta que no el programa llegue a la última fotografía (no hace falta que se descarguen todas, si alguna da error el archivo se creará igualmente).

A.1.2. Cargar los datasets

Como hemos dicho anteriormente esta función no tiene utilidad ya que no se hace nada con los datos, existe simplemente para probar y comprobar que se cargan bien. Al pulsar la opción 2 en el menú principal nos va a preguntar qué *dataset* y que conjunto queremos cargar:

```
Choice: 2
Enter the name of the dataset to load (coco, flickr, bing): coco
Enter the name of the set to load (train, validation, test):
    validation
```

Esta función nos puede devolver tres posibles mensajes:

- `The selected dataset is not available`: Este mensaje indica que el *dataset* seleccionado no se encuentra disponible, es decir, que no se ha descargado.
- `The dataset has been successfully loaded`: Este mensaje nos indica que el *dataset* se ha cargado correctamente.

- `There was a problem loading this dataset`: Este es el mensaje de error que se mostrará en caso de que sí que exista el *dataset* elegido pero haya habido algún problema al intentar cargarlo.

Ahora pasamos a explicar como se guardan los datos.

A.1.3. Formato de los datos

Para guardar los datos se va a crear una carpeta llamada `data` dentro de la carpeta `datasets` que es la que contiene todo lo relacionado con los *datasets*. Esta carpeta a su vez se divide en varias carpetas con el nombre del *dataset*. Dentro de cada carpeta de *dataset* va a haber otras tres carpetas con el conjunto de datos que contienen. Por ejemplo, si queremos acceder a las imágenes del conjunto de entrenamiento de `validation` del *dataset* `coco` tendremos que ir a `datasets/data/coco/validation`.

Las imágenes se van a guardar en formato `.jpg` y su nombre va a ser el nombre del *dataset* y su id, `<nombre_dataset>_<id>.jpg`. Por ejemplo, al guardar una imagen del *dataset* `coco`, esta imagen se guardará como `coco_78087.jpg`.

Para las preguntas vamos a crear un archivo *.json* con nombre será el propio del *dataset* y del conjunto, `<nombre_dataset>_<nombre_conjunto>_questions.json`. Por ejemplo, cuando se terminen de descargar todas las imágenes del *dataset* coco y el conjunto de `validation` se creará en el directorio del *dataset* el archivo `coco_validation_questions.json`. Vamos explicar como se guardan las preguntas en este archivo `.json`. Este archivo es en realidad una lista de objetos *JavaScript Object Notation* (JSON) con el siguiente formato:

```
{
    "id": "coco_314392",
    "questions": [
        "What is she drinking?",
        "What are you drinking?",
        "What is her name?",
        "What is this person drinking?",
        "Who is she?"
    ]
}
```

Como vemos cada objeto contiene dos identificadores. El primero `id` contiene un *string* con el identificador de la imagen. El segundo `questions` contiene una lista con todas las preguntas de esa fotografía.

Cuando vayamos a utilizar un *dataset* podremos cargar tanto las imágenes como las preguntas llamando a la función `load(<dataset>, <set>)` que cargará el conjunto de datos *set* del *dataset* `dataset`. Esta función devuelve dos listas `questions`, `images`. La primera contiene una lista de objetos con el mismo formato que el JSON explicando anteriormente (la propia función transforma el JSON a objeto). La segunda lista contiene objetos con el id de la imagen y la imagen en si. El formato es el siguiente:

```
{
    "id": "coco_314392",
    "image": image
}
```

El conjunto de datos *mscoco* descargado de forma manual desde internet no tiene el mismo formato que el resto de datos que hemos explicado anteriormente por lo que se ha creado una función específica para que se guarden todos los datos de la misma forma. Una vez incluido el archivo JSON solo tendremos que ejecutar la tercera opción del menú y generará el conjunto de preguntas bien formateadas[1].

A.2. Manual de uso del módulo Vocabulary

Este módulo está pensado para ser de fácil ejecución. Una vez tengamos todos los datos descargados y formateados y presentes en sus respectivos directorios, podremos ejecutar este módulo.

Este módulo está dividido en dos partes, aunque al ejecutarlo simplemente tendremos que pulsar el botón ejecutar dentro de nuestro entorno de programación. Una vez ejecutado podremos ver en el directorio `/source/datasets/vocabulary` dos archivos con el vocabulario completo y reducido. Esto significa que se ha ejecutado correctamente.

[1] Notebook con todos los detalles sobre el módulo dataset: https://colab.research.google.com/drive/1QIbcT9m3kzxDj2BkIuYnGkqvg3z5AVNh?usp=sharing

Apéndice B

Instalación del Bot

En este apéndice vamos a explicar como instalar el *bot* en nuestra aplicación de Telegram. Tenemos varias opciones. Para ambas opciones necesitamos tener la aplicación de Telegram descargada, no es válido para la versión web.

B.0.1. Opción 1

Esta opción es quizás la más sencilla ya que solo tenemos que pulsar en la imagen que se encuentra arriba o introducir el siguiente enlace en nuestro navegador:

https://t.me/Remi_Therapy_Bot

Este *link* nos llevará a una página web y nos preguntará si queremos abrir el *link* en Telegram. Le daremos a Permitir y ya tendremos el bot listo para utilizar.

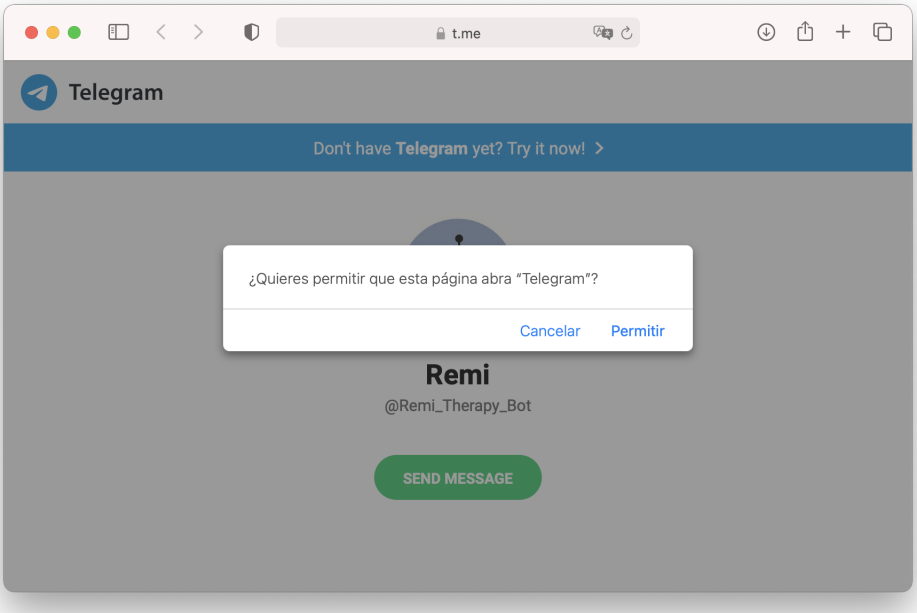

Figura B.1: Click en el link

B.0.2. Opción 2

Esta segunda opción consiste en Buscar a Remi en la barra del buscador. Para ello solo tenemos que buscar **Remi_Therapy_Bot** en la barra de búsqueda y seleccionar la opción que aparece.

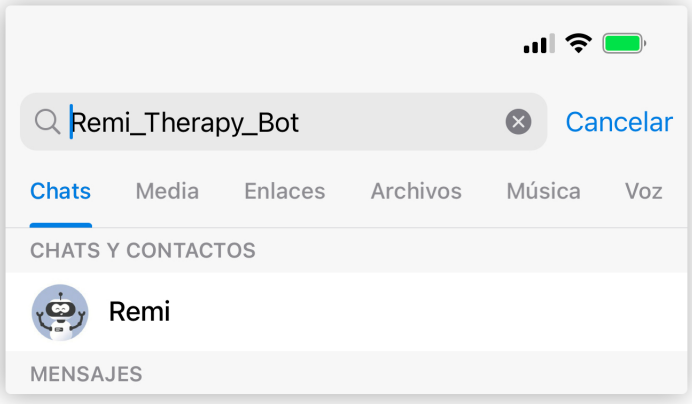

Figura B.2: Click en el link

Una vez hagamos click al bot ya podremos comenzar a utilizarlo.

Bibliografía

[1] Mariona Caros, Maite Garolera, Petia Radeva, and Xavier Giro i Nieto. Automatic reminiscence therapy for dementia, 2021.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[3] Yikang Li, Nan Duan, Bolei Zhou, Xiao Chu, Wanli Ouyang, and Xiaogang Wang. Visual question generation as dual task of visual question answering, 2017.

[4] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[5] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality, 2013.

[6] Andrew Ng. *Deep Learning Specialization*. Coursera, 2017.

[7] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[8] Mariona Caros Roca. A generative dialogue system for reminiscence therapy. Master's thesis, Universitat Politècnica de Catalunya, 2019.

[9] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[10] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[11] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.

[12] Sam Wiseman and Alexander M. Rush. Sequence-to-sequence learning as beam-search optimization, 2016.

Siglas

API *Application Programming Interface*. 26, 45, 50, 52

BD *Base de Datos*. 6, 46–48

CSV *Comma Separated Values*. 74

CV *Computer Vision*. 14

DL *Deep Learning*. 8, 10–12, 14, 16, 28, 29

IA Inteligencia Artificial. 11, 26

JSON *JavaScript Object Notation*. 77, 78

LSTM *Long Short Term Memory*. 18, 38

ML *Machine Learning*. 11, 12, 26

PDF *Portable Document Format*. 27, 43

PLN Procesamiento del Lenguaje Natural. 20

RGB *Red Green and Blue*. 14

RN Redes Neuronales. 12–14, 16–18, 20, 26, 27, 29, 30, 37, 40

RNC Redes Neuronales Convolucionales. 12, 15, 16, 21, 36, 39

RNR Redes Neuronales Recurrentes. 13, 16, 17, 19, 22, 36, 37, 39

SGBD *Sistema Gestor de Bases de Datos*. 46

SQL *Structured Query Language*. 46

VQG *Visual Question Generation*. 8, 11, 21, 23, 30, 31, 71